FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Marek Židek

Controlled Music Generation with Deep Learning

Institute of Formal and Applied Linguistics

Supervisor of the master thesis: Mgr. Jan Hajič, Ph.D.
Study programme: Computer Science
Study branch: Artificial Intellingence

Prague 2020

I thank my parents for their support during writing this thesis. My mother for her care and affection. My father for his love and support during all my studies. I thank my girlfriend for her patience and positive attitude. I thank my supervisor for his endless ideas, for his time and dedication to help. I thank the teachers at the faculty who taught me valuable knowledge in respectful and professional ways.

Title: Controlled Music Generation with Deep Learning

Author: Marek Židek

Institute: Institute of Formal and Applied Linguistics

Supervisor: Mgr. Jan Hajič, Ph.D., Institute of Formal and Applied Linguistics

Abstract: Generation of musical compositions is one of the hardest tasks for artificial intelligence where most of the current approaches struggle with long term coherence of the generated compositions. This work aims to demonstrate how deep learning models for generating music can be externally controlled to produce compositions with long term coherence, polyphony, and multiple instruments. We work with classical music ranging from compositions for piano through string quartet and up to symphonic orchestral compositions. To control the generation process, we take inspiration from the abstract notion of musical form: normally a high-level description of how similar and dissimilar passages are arranged throughout a composition, we use it as a recipe for generating a coherent composition. To this end, we (1) design a sufficiently general music similarity pseudometric from existing methods, (2) extract musical form from the training data by applying a clustering algorithm over the similarity values, (3) train three models that generate similar and locally coherent dissimilar musical fragments, and (4) design a way how to use the musical forms during the generation process to orchestrate the inference of the three models to generate whole compositions from musical fragments. We show what is the performance of the transformer generative models for generating musical fragments when trained by presenting one-to-many training examples. Multiple runs of the variant generating model given a same single input fragment can be used as a palette for variations inspiration to the human composers. To evaluate the results of the generation system, we experimented with adaptation of the Fréchet inception distance for music, which suggests that the generator-produced compositions are in some sense close to compositions by human authors. However, the truly relevant evaluation is whether the system has been able to produce compositions engaging for human listeners (or performers). Based on the generated samples, we believe that it has had some success, but we encourage the readers to judge for themselves.

Keywords: Music generation, Deep learning, Computational creativity, Machine learning, Music similarity

Contents

Introduction

Composition of a musical piece is one of the highest demonstrations of human creativity. This poses an interesting problem for artificial intelligence. There has been a great advance in the field of generating musical compositions thanks to the use of Deep Learning techniques, e.g. convolutional or recurrent neural networks, or more advanced models, such as variational autoencoders (VAEs) or generative adversarial networks (GANs).[1] These advances can help with imitating musical creativity and be used as tools to help composers with their creations either by inspiring them or by elaborating the musical material. A major shortcoming of current approaches to both symbolic or raw waveform audio is the long-term coherence of the generated pieces, which should generally be composed of recurring sections with a certain hierarchy.

In this work, we explore a hybrid approach for automated music generation in a *more controlled* manner. The term “control” refers to the fact that the neural network models usually generate music as black boxes and the user has no influence over various musical properties of the result. In our work, control means not trying to train the generative model to implicitly capture long term dependencies, as the current machine learning methods struggle to do so. These dependencies are taken care of by a different algorithm that uses domain knowledge and performs an analysis to obtain long term relations viewing the whole composition as assembled from smaller musical fragments.

The approach this work chooses is to explicitly generate repetitions, instrumental accompaniments, variations, or transitions between motives. The key factor in making a well-received composition is creating a good balance between the forces of surprise and expectation. This means keeping the attention of the listener by adding new musical material against the satisfaction the listener gains from listening to a familiar material. The layout of familiar versus unfamiliar can be used as the most abstract definition of what a *musical form* is.

To make use of such a form, we have to be able to generate similar, dissimilar and locally coherent fragments of music given an input fragment. The local coherency means that the generated musical fragments should fit well together when concatenated in an order described by the musical form and have good inter-instrument/track complementarity when the individual voices/tracks are put together as described by the musical form. We work with unrestricted polyphony, ranging from compositions for a piano to an entire orchestra.

Getting such a form is another task because it is not explicitly marked in training compositions. It could be extracted from the data or in an unsupervised fashion, or labeled by experienced analysts. We chose to extract the forms from the data by an algorithm. The core problems for the extraction algorithm is that the resulting form should correspond to *perceptual* similarity of the fragments and that the algorithm should be able to find out if two fragments are similar enough to be considered as *variations* on the other. Perceptual similarity is a very hard concept as it depends on both objective and subjective criteria and algorithms

[1]It is expected that the reader has a base knowledge of concepts in e.g. Goodfellow’s et al.: Deep learning, 2016 [1] or other book or course, however, most of the core concepts will be defined in this work

cannot find similarities that happen on the subconscious level in the minds of listeners with various experiences and states of mind. However, there are certain helpful concepts from music theory that are already formalized and are the best approximation of perceptual similarity.

This processing of the data then allows us to obtain training examples of a reasonably short length (e.g. measures) that are fit for *sequence to sequence* (seq2seq) models that could generate variation, accompaniment and new musical idea measures (that are nevertheless continuation of the previous measures), and at the same time to have a straightforward way of combining the generated fragments together to create the final composition.

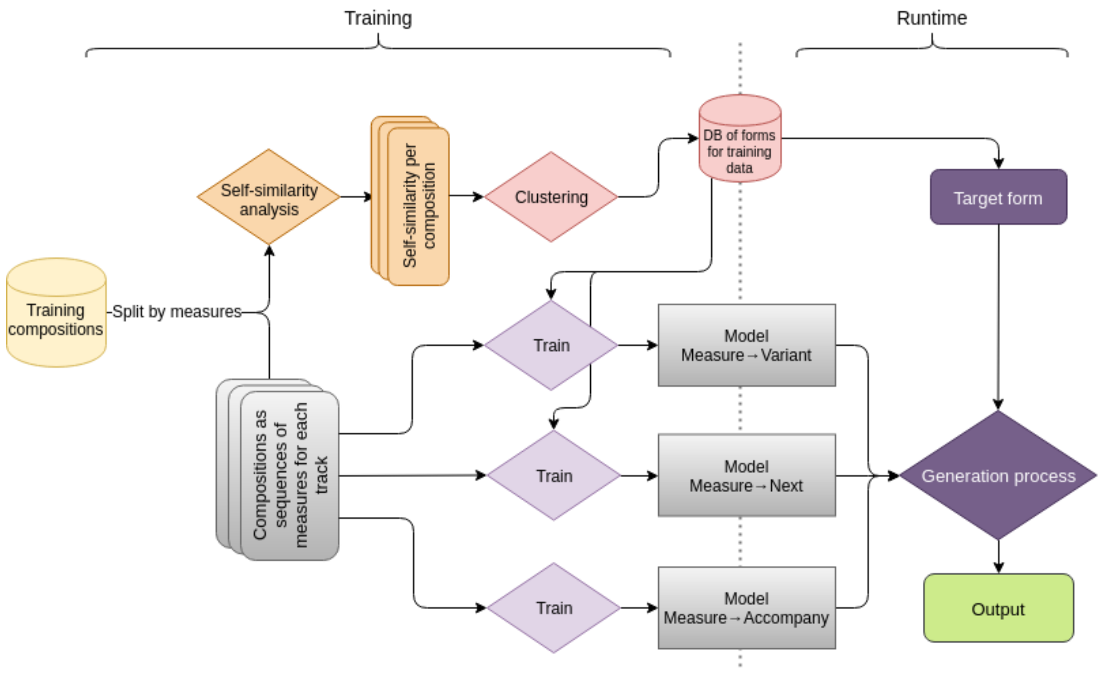


Figure 1: A simplified workflow of the whole process of our music generation. The left side of the diagram describes the steps of the process that prepares and trains the components of the generation system, the right side describes, in simplified terms, the process at runtime. The colors in this diagram follow the structure of this work. Yellow is for Data chapter, 4, orange is for Similarity Model chapter 5, light red is for Form Extraction chapter 6, light purple is for chapter about the generation models 7, dark purple is for chapter Generation Scheme 8 and the green color is for the Generation Results chapter 9.

In Figure 1, we can see the overview of the whole generating system. The training compositions are analyzed by self-similarity analysis and the similar fragments of the compositions are clustered together to extract a musical form. The clusterings provide training pairs for a model that generates variants (fragment pairs that are assigned to the same cluster), and conversely indications of what training pairs to avoid for the model that generates non-variants. Then, the form is used alongside with the two models and a third, accompany generator, model to generate fragments in a not necessarily "start-to-end" order that are concatenated together as the extracted form describes.

We work with a similarity metric for symbolically represented polyphonic music mainly based on [2] and [3]. Our analysis looks into intra/inter instrumental similarities between measures to obtain self-similarity matrix for each composition.

Finding the fragments similar enough to be considered variants of each other is done through clustering and adjusting the clustering algorithm hyperparameters through an automated search for each composition according to our goals, the overall level of intra-opus homogeneity differs among compositions, but within each (good) composition there is a certain amount of surprise.

Having the similar fragments clustered, we train models for generation of variant fragments, non-variant continuation fragments and, accompany fragments.

With the clusters of similar fragments, we can extract a musical form that describes what fragments should be similar and where should they be placed in a resulting composition.

How do we know our system works? Evaluation of music generation is a difficult task which, for the time being, takes place mostly on the qualitative levels. We cannot expect rigid evaluation as in more objective and longer addressed tasks (machine translation). This can be supported by the related work which usually rely on subjective evaluation or questionnaires. To evaluate our generated results, we look into the methods from image evaluation of generative models [4] and try to adapt them for music.

The resulting system cannot of course be expected to produce pleasant music on every try. Rather, we are interested in exploring whether our approach that, as opposed to other works using machine learning, explicitly uses the musical form can be a concurrent to the “traditional” machine learning generation approach or whether it has a chance to produce pleasant music.

This work explains our new hybrid approach and shows that it is able to produce long coherent multitrack imitations of classical music. The individual models used in our system have a potential to be also used ad-hoc to provide inspirational fragments of music in a process of human composition.

The thesis is structured by chapters as follows: We first introduce the basics of musical terminology, then continue with our goals and system design, look into the related work and our dataset. Then, we explain the three main blocks of our approach in the next three chapters, i.e. the similarity comparison model, process of musical form extraction and deep learning models for generating fragments. Then, we look at how we generate the whole musical compositions buy using musical form and the three deep learning models. Finally, we show our resulting compositions and an adaptation of a known evaluation technique for image generation and conclude this work.

1. Brief Primer on Musical Terminology

In this chapter, we introduce in more detail the problem and the goals as approached in this work. We first have to formalize the notion of music and musical form, and then we can re-state the goals of this work as actionable steps.

Music is a way of organizing various sounds into time that holds some meaning and evokes emotions. Pleasant musical pieces usually follow rules of *harmony*, *melody* and *rhythm* to some degree. Music is also usually well structured in its contents, meaning there are repetitions and elaborations of themes put well together. However, there are also non-descriptive elements such as expectancy vs. surprise ratio, frustration, tension, or giving a metaphorical meaning or emotion.

We will describe music solely as a temporal arrangement of *notes*. A note has 4 main attributes that are *pitch*, *duration*, *strength* and *timbre*. Pitch is a frequency given some *tuning*[1], duration is a time span for which the note should be performed, strength is loudness, and timbre describes spectral characteristics and additional noise [6], [7]. Pitch and duration are creating the base of a composition, while strength and timbre are more about the performance of the musical peace and could be left to the performing musician to interpret them. Following the definitions in [7], pitch is specified as:

> *"Pitch is a categorical variable with values from a subset of frequencies defined by scale and tuning. For western music, the scale of available pitches consists of semitones, using a tuning system where the next higher semitone is determined by the previous 1 as* $s_{i+1} = s_i \cdot 2^{f\frac{1}{12}}$*. Pitches are ordered linearly by their ascending frequency and arranged into a repeating pattern (i.e. C, C♮,D,D♮,...) of 12 notes called semitones. One repetition of this pattern is called an octave. Octaves are numbered from 1 to 8, in the English-speaking tradition. The number of octave is usually concatenated to the name of note (e.g. C5). For a base octave and a note, the same note in any other octave can be derived by stacking multiplication or division of the note's frequency by 2 (e.g. midi tone 48 (C4) and 60 (C5), are both on step "C", because the C5 pitch frequency is exactly twice the C4 frequency. That also means they resound a perfect harmony). In MIDI format, the pitches are numbered by integers in ascending order by their frequency."*

and duration is specified as:

> *" Duration is also categorical*[2] *. Duration is measured in beats (units of musical time). The duration categories are fractions of a beat, mostly in powers of 2. In musical terminology a 4-beat note is called a* whole *note, a 2-beat note is a* half *note, then* quarter, eight *etc. The most*

[1]Tuning is a way of determining a frequency of a given pitch given f reference tone and formula to calculate other pitches [5]

[2][7]: Categorical in music theory, although its realization is not: musicians slow down or speed up the flow of musical time in order to bring out inner structures in the music"

common grouping of beats into regular units (called bars/measures) *is by 4, hence the name "whole". However, durations that do not fit into this "powers-of-two" (e.g. 1/3 or 2/5) are not an exception in most of the compositions."*

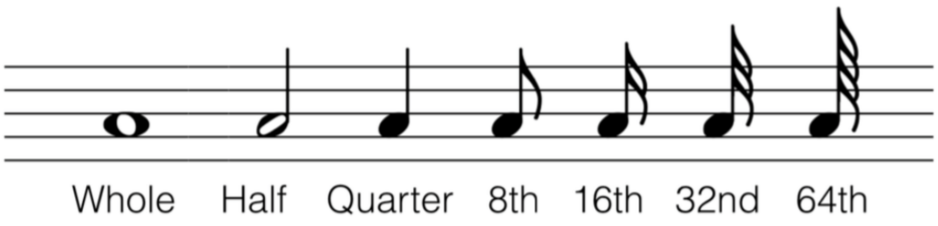


Figure 1.1: Types of notes according to their duration[6]. The lowest duration we consider in our generating models is the 64th.

Term *rest* is used for silent moment in composition or for a given instrument, while the other instruments can be playing. It has the same duration specifications as a normal note, but it has no pitch, no strength and no timbre. We use a term *rest-measure* to describe a measure which only contains rests.

Throughout this work, we are going to work with discrete symbolic representation of music. The approach of formalizing music as a collection of abstract note objects rather than the sounds they imply is called the *symbolic* approach in music generation literature. A disadvantage of the symbolic approach is that the synthesis of the sound is static, so there are no sound effects and timbre changes except the definition of the *instrument*[3] to play along with its discretized loudness. However, we are trying to generate the compositions and not their performance so we avoid the domain of soundwaves representation which can have lots of noise and is has much higher sampling rates for our models to be able to cope with longer sequences.

1.1 Time in Music

Time in music is counted through pulsing units called *beats*.

Meter organizes music in time through regularly recurring pulses called beats in a musical bar/measure featuring different patterns and accents through the measures. The beats are not needed to be sounded, but are still expected and perceived by the listener.

Basic classification of meters are simple and compound meters, and duple, triple and quadruple metes [8].

The classification of simple and compound captures the relationship between the counting pulse (a beat) and pulses that are faster than it. It could be said that they divide the beat, where simple meter divides the beat into two equal parts, whereas compound meter divides the beat into three equal parts [8].

[3]Instrument is used throughout the work as a short for musical instrument, where context should help to disambiguate when instrument is used in different meaning

The duple, triple, quadruple classification captures the relationship between the counting pulse and pulses that are slower than it. That is they group the beats into bars/measures, where duple groups into two, triple into three and quadruple into four.

The time signature is noted by two numbers at the beginning of a starting measure, where the top number determines the type of meter (2,3,4 - simple duple, triple, quadruple, and 6, 9, 12 - compound duple, triple, quadruple), and the bottom number determines which type of note (their duration) corresponds to a single beat in simple meters and to a single division of a beat in compound meters. The beats are usually alternated as stressed and unstressed pulses.

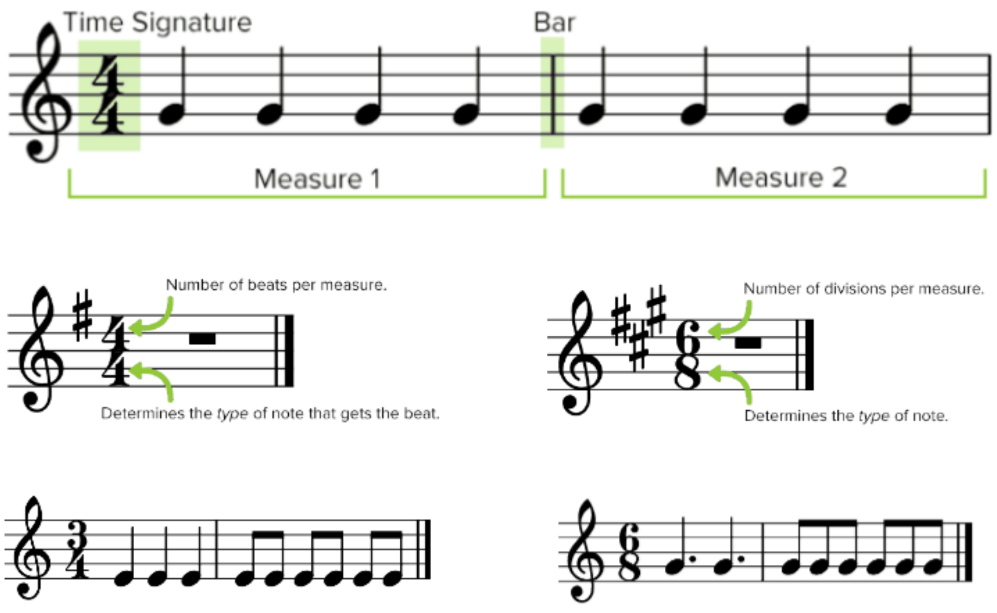

Time Signature
Bar
Measure 1
Measure 2
Number of beats per measure.
Determines the *type* of note that gets the beat.
Number of divisions per measure.
Determines the *type* of note.

Figure 1.2: Time signature and measures with examples of simple and compound meters. Individual figures are taken from [9]

Rhythm is the placement of sounds in time and depends on the meter and *tempo*[4].

Downbeat is the first beat of the measure and is usually stressed as it is a beginning of the sound representing musical energy going in a forward motion. *Upbeat* is the last beat of a measure and prepares the listener for the downbeat creating a sense of anticipation [10]. We will also use the *onbeat* in some meters (e.g. simple quadruple), which is the second strongest accent after the downbeat in the melody and a second most likely place for a chord change. The onbeats are usually it the middle of a measure. Other beats that are subdivisions of the main beats are usually weakest and are sometimes called offbeats.

[4]Tempo can be considered as a relation of beat and time, practically speaking, it is the speed of the composition for a given composition or a part of a composition

1.2 Key

A *scale* is a set of pitches in ascending or descending order with specification (*mode*) in a number of half-steps for each successive pitch from the previous pitch. *Key* defines a group of pitches or a scale and its first pitch called *tonic*. We will work with two main modes familiar in Western classical music, *Major* and *Minor*. They have 7 tones in intervals +0, +2, +4, +5, +7, +9, +11 for the Major, +0, +2, +3, +5, +7, +8, +10 for the Minor. Every tone in the key sounds more or less consonant with the others from scale [7].

A part of music written in a given key uses mostly the set of pitches determined by it but can violate it from time to time to create moments of surprise.

Musical compositions are typically written in one key though the key can change throughout the composition e.g. *Zd. Fibich. Sonatina. Op. 27.*.

1.3 Intervals

Intervals are differences in pitch between notes that are counted in semitones (in this simple explanation, we consider only *dyads* of two notes sounded together). These intervals are important because we perceive two or more simultaneously sounding pitches as of ratios of their frequencies instead of distinct sounds. Some ratios sound pleasant, harmonic or *consonant*, where some are *dissonant* and not pleasant to our ears. A clearer definition of consonance is more complex and can be found in [5]. However, the perception of a consonant sound is relative to the listener and their culture/preferences and can also be framed by preceding notes, where a dissonant sound may be heard as consonant if it is preceded by many sounds that are even more dissonant [5].

Interval names are dependent on their scale, however, we will work only with chromatic intervals (counted in semitones).

In western classical music, consonant intervals are considered (as degrees for a given scale) 1, 3, 5, 6, 8 and dissonant intervals are considered 2, 4, 7. Throughout most of the history of polyphonic music, the basic principle in music is organisation of harmony with appropriate alternation of consonance and dissonance.

Compound intervals are simply intervals that are larger than 12 semitones, however, they are usually treated as "modulo 12" and considered as on similar level of consonance as their simple counterparts.

Two intervals are *inverse intervals* if they have their pitch classes (pitch modulo 12) the same and the sum of both chromatic intervals is 12.

Intervals can be *melodic* or *harmonic*. The interval between the pitches sounding at the same time is called harmonic. If the two pitches are sounded one after another the interval between them is a melodic.

1.4 Chords

A chord is any combination of three or more pitch classes that sound simultaneously. There are many versions of pleasant chords defined by their root note and a quality(intervals between the three notes), which are defined for 3-notes chords and 4-notes chords.

An important concept related to chords is the *arpeggio*, which is a chord that is broken into a sequence played in a quick succession in usually ascending or descending order, or even other permutation (even permutation with repeat) of (usually consonant) notes belonging to the chord.

1.5 Variation Techniques

Variation is technique where musical pattern is repeated in a slightly altered manner where we could still recognize it. The changes may involve combinations of changes in rhythm, melody, harmony or orchestration. Other changes are *inverting the melody*, *retrograde* (reversing the melody) and the perceptually least confounding change is *chromatic/diatonic transpositions.*

Inverting a melody means that the steps in melody that are ascending a specified interval become descending by the same interval in the inverted melody and vice versa. Retrogrades are melodies played from the end to the start, either only pitches or with rhythm as well. Transpositions (or *contours*) are simple shifts of the whole melody over some interval could be be either chromatic and diatonic, where diatonic means that the steps to shift for each note are determined by the scale.

1.6 Non-mathematical Relationships in Music

Even though certain dimensions of a musical piece can be analyzed by more or less formal theory, there are some conscious or unconscious mind processes required from the composer to complete a composition [11]. As written in [11]: *"The consciously articulated rules of even the best-understood musical styles have not shown themselves to be sufficient to capture satisfactorily the underlying compositional process."* and *"It is generally the case that composers are more interested in establishing a formal device as a point of departure than as an end in itself."*

To create a sense of tension or emotion to keep the listener engaged, harmonies, cadences, dynamics or rhythms are alternated with less defined harmonies or rhythms, more pronounced cadences or different dynamics or even complete dissonances. One example might be repeating a sequence two times and a half and then switching the harmonic resolution, where repeating the sequence twice creates a sense of expectation, twice and a half confirms the expectation and creates a temptation to stop paying attention, and a sudden breaking of a pattern creates a deceit of expectations and sustains the listeners interest.[11]. The theory [12] behind why it works is that consonance and dissonance are a function of centrifugal and centripetal forces in music. These forces are described [12] as: *"In Schoenberg's context, centrifugal forces are those that require expansion, like Phiolaus's unlimiteds, they constitute the potential for development with a musical idea. Centripetal forces are those that lead to coherence, they hold the idea together and make us perceive it as a unity."* However, the tension from the theory can be achieved through the sole use of timing (solo drum), dynamics, timbre, cadence, or spectral changes of the sounds (electronic music). In a musical mind, there are certain association networks formed such that given a partial input, the pattern completion mechanisms consider many possible continuations and a composer, who understands this predictive mechanism can shift these expectancies to their own aim [11]. Therefore, we are looking for a computer model, which is able to tackle this critical aspect.

1.7 Musical Form

Musical form is defining the analytic structure of musical composition and the syntactic rules relating to it.

There can be more or less defining organizational elements along with their hierarchical organization. These are temporal layouts of melody, harmony or rhythm that are repeated and varied by some degree of similarity. They could also be be defining intonation, density of notes, dynamics, choice of instruments, orchestration or other musical units [13].

Karl Eschman has also written on explanation of what a form is: *"There remain the two different requirements for the production of any form: some differentiation of materials, and some relation of these differentiated materials by repetition."* [14] p.23.

1.7.1 Phrases and Motives

A *phrase* is a unit of the musical form which has a complete musical sense on its own, [15] [16]. Phrases are usually not of a predefined length, but they can be around four measures long. Phrases are consisted of smaller units called *motives* and *figures*. Motif is a short sequence of notes that occurs recurrently and defines some character or theme to a composition, where figure is fairly similar to motif but carries less significance to the whole composition and can be endlessly chained together in the background [16], however, it could be that a composition is created only from figures e.g. minimal techno music. Both motives and figures are not of a predefined length.

One of the main requirements of the musical form is to group the same or similar phrases in music and define their arrangements into time along with some more or less defined alternations. In order to extract or analyze a form, one has to have a notion of how the similarity of musical pieces is perceived and at least partially "formalized" as musical similarity can be subjective.

2. System Design

Our main goal is to generate long, multi-instrumental and coherent musical compositions. Current models that look at music generation as a monolithic machine learning problem [17], [18] are unable to maintain coherence for longer than 2000 tokens (60 seconds) [19].

The core concept of our approach is to utilize an explicit musical form (see Sec. 1.7), which is the outline of a long-term structure of a composition on various time scales by defining repetitions and similarities of its parts. It follows that with a help of a musical form and models that are able to generate musical units that follow the “template“ or recipe for similarity and dissimilarity provided by the form, we should be able to generate longer coherent sequences. Two further types of local coherence need to be maintained: temporal, where the “next“ new part should relate to what was heard previously (possibly by being suitably different from it), and harmonic coherence in polyphonic compositions, which should ensure that the voices which sound together accompany each other well.

However, such an approach presents new problems since the musical form as defined in music theory is not sufficiently formal for a computational application, from which follows that form annotations are not present in datasets. We therefore will have to define and then extract musical form from data ourselves, based on the concept of similar and dissimilar parts. However, a function that evaluates the similarity of two polyphonic fragments of music is also not readily available.

Therefore, in order to apply musical form for generating long-term coherent compositions, the following steps are necessary:

1. Design of a musical similarity pseudometric
2. Calculation of self-similarity matrix for a given composition
3. Clustering of parts of composition that are similar enough to be considered variants of each other
4. Extraction of musical form from a given composition
5. Training of a model which is able to produce variants of a given motif measure
6. Training of a model which is able to produce a measure that is continuous to the previous measure(s[1])
7. Training of a model which is able to produce a measure for other voices played in parallel
8. Design of a generation process that uses the 3 previous models in a meaningful way to produce a desired output

These processes can be seen in Figure 2.1 where the colors indicate the chapters of this work corresponding to the processes that are necessary.

[1]One can anticipate that the previous measure might as well be a “rest measure” which gives no real indications of how the track should continue

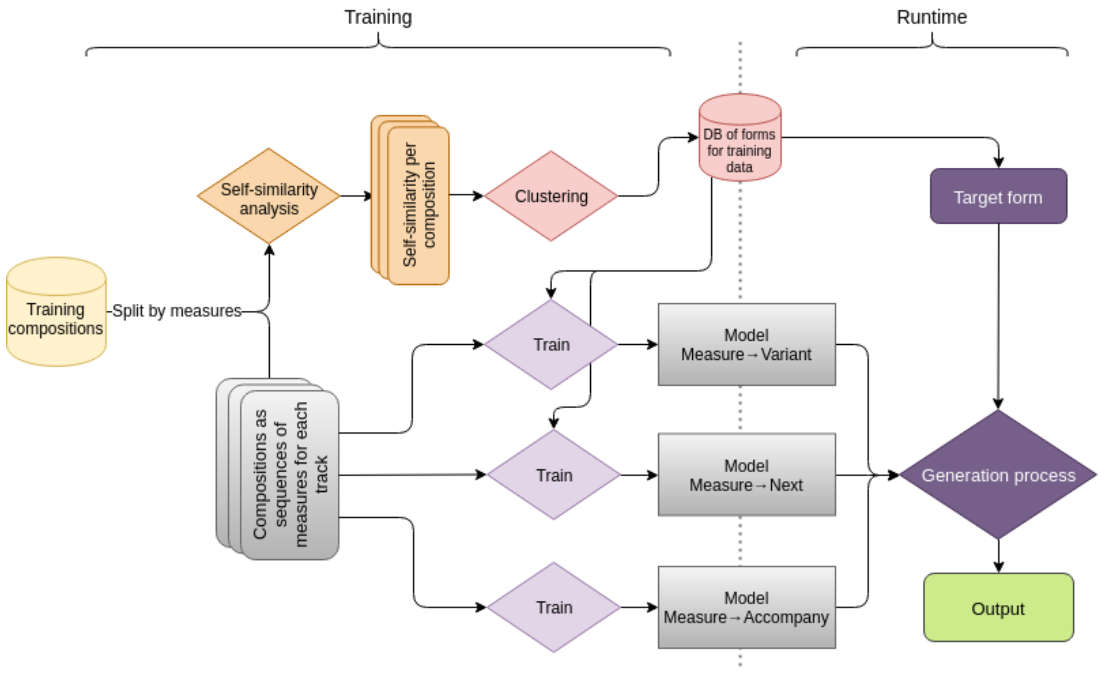


Figure 2.1: A simplified workflow of the whole process of our music generation. The left side of the diagram describes the steps of the process that prepares and trains the components of the generation system, the right side describes, in simplified terms, the process at runtime. The colors in this diagram correspond to the numbers of processes in the previous list. Orange is for (1) and (2), light red is for (3) and (4), light purple is for (5), (6) and (7), and the dark purple colour is for process number (8).

2.1 Similarity within Musical Pieces

Musical pieces or sections of compositions can be compared from various different perspectives including name, author, key, instrumentation or by musical content comparing domains like pitch, rhythm, harmony, texture, meter, tempo, timbre, dynamics etc. The similarity information can be obtained from raw data or from symbolically encoded data. We, as throughout this whole thesis, work with symbolic music.

2.1.1 Features important for similarity

The most important attributes from the musical domain for comparison of musical pieces are pitch, pitch contours and rhythm [20]. Many of the known frameworks for musical comparison use just these two attributes, e.g. in [21] or [2]. Another important factor is the requirement to find similarities regardless of the position in measure, interactions with other patterns or different lengths of the patterns.

Many comparison systems [2], [22] or [23] take into consideration only monophonic patterns, whereas many compositions have some degree of polyphony within their tracks. Monophonic music is to be composed of one melody i.e. that only one note is sounded at a given time. In contrast, polyphonic music can have more than one note sounded at a given time and there is no perfect technique to extract melody from polyphonic music [3], yet, some approaches try approximation of a monophonic reduction and then use a monophonic comparison systems. When working with polyphonic music, it means that we need to find similarities of chords, and the chords can even by similar to several consecutive notes (arpeggio) [3].

2.1.2 Goal

The goal is to have a similarity pseudometric usable for internal analysis of polyphonic compositions of different lengths. The desired output is either to have ordered sequence of triples of: bounding interval for first subpart, bounding interval for a second subpart, and their (dis)similarity value. These would be ordered from highest to lowest similarity, or a self-similarity matrix of predefined comparison lengths (e.g. 1 measure). In case of multiple simultaneously playing tracks which could have different instruments, we want to compare the tracks separately so that any part in any track is analyzed against all the other parts to analyze cross-instrumental similarities.

2.2 Musical Form Extraction

The extraction of musical form requires finding the perceptually similar and dissimilar fragments of music, grouping them together and keeping their arrangements in time. One can even keep some context as a part of a form, such as instrumentation, scale/mode of a key, transpositions, dynamics, motion etc.

Problem with grouping the similar fragments within musical part can be also highly subjective. Tunes can be similar to one another, by some respect, but not variants of each other [24]. As Meudic, Benoit [20] *p. 2.* states: *"Another problem arises from temporal considerations: should a similarity algorithm be independent of the context? This is hardly the case when considering that our culture, education and memory influence our perception of similarity. However, one can still wonder if there exists some universals."* Here, context could also be other notes in the composition preceding the analyzed part. Other context such as rhythm, scale etc. are, however, already taken into consideration by the similarity pseudometric.

Apart from similarity subjectivity, another problem lies in the definition of how the groupings (clusters) from the extraction algorithm should look. In ideal case, the vision of a form in human composer's mind is bound to match his/her compositions musical form according to the composer's hierarchy in their similarity perception. However, an algorithm that works in reverse, trying to get into the composer's mind, needs to be instructed on structural characteristics of the desired groupings. Such characteristics are different for every composition and they could be:

The proportion of fragments in a cluster against the number of fragments in the composition

Inner structure of the clusters, such as meaningful variances of intracluster similarities.

Differences in cluster sizes and densities of fragments inside clusters

Number of clusters in total

2.2.1 Our Form Definition

Our base definition of a form is a simplification in which we consider only parts of a given arbitrary absolute duration, in our case one measure (the absolute duration is dependent on a meter). For purposes of this introductory chapter, the most basic version of our form consists of tuples (P, V, T, N) arranged as of their order in the composition. P stands for *primary symbol*, which determines the class of the similarity grouping, V determines the numbering of a variant within a given grouping primary symbol, T determines chromatic transposition (no transposition and a same variant implies identity), N stands for track number in an ordering given by analyzed composition.

2.2.2 Clustering the Fragments

For grouping of the similar patterns, we experimented with various clustering algorithms. Some of them required to predefine the number of clusters or required projecting our similarity metric into a linear subspace (though our final used similarity is not strictly a metric, the triangular inequality is not satisfied and 0 similarity value is not equivalent to the compared sequences to be the same). We worked with precomputed similarity matrices and determined the number of clusters through the clustering algorithm parameters to meet our predefined goals as specified by a criterion. Another concern when grouping the variants is the question of context. Is the similarity of pieces sounded in a middle of a composition perceived differently when primed by listening to a same composition for e.g. 2 minutes?

2.2.3 Goal

The goal of the extraction is not only to get the form, but also to collect training data for two models. A model which would be able to generate variations trained on data which are put into a same cluster during extraction. And a model that generates non-variants trained on data which are not put into a same cluster. By first extracting the form and then training the models to provide the parts predefined by it, we can make use of the extraction for another time and just put the generated parts into a time axis as defined by ordering from the extracted form. The extracted form should specify what parts of the composition are same, which are only transposed, and which are similar or dissimilar disregarding the true value of a similarity between them. We also want to encode more things into our musical form definition that are used for conditioning our models, such as instrumentation, meter, track number etc., however, these are not hard to extract from the training data from the algorithmic point of view. Other attributes for such conditioning are density, average velocity, degree of polyphony or general motion in context of the part of the composition and its surroundings.

2.3 Measure Generator Models

Typical scheme for generating a musical piece with the help of machine learning algorithms consists of training typically a sequential model and then, during generation, priming it with a starting sequence from training data and then letting it feed its predictions back into itself, continually generating the whole new musical sequence.

As suggested in the introduction, we are going to use our generative models to provide us with only fragments of the whole composition instead of sequentially generating the composition as a whole. For such approach, we are going to use sequence-to-sequence models. These models given a sequence generate a new sequence conditioned by the first.

We will describe how we can encode music in computers and how we can represent it for our models as there can be many representations available [25].

2.3.1 Representation

We will represent symbolic pieces of musical composition as sequences of events as in [26], which allows the models to learn expressive timings with granularity of 10 milliseconds and loudness of the notes. While we do not work with expressive timing, this representation has advantages: e.g. it encodes the music similarly to how sparse matrices are encoded which allows for higher informational value, and it “hides” the polyphonic dependencies not easily trainable due to an combinatorial explosion into a temporal domain. Our events are one-hot encoded vectors of possible 88 note-ons, 88 note-offs, 64 duration-shifts and 16 *velocity* (loudness) bins. If we add *SOS*, *EOS* and *padding* events we have 259 different events. However, we will also have many context variables which will be needed to add to this representation e.g. scale, motion, instrument, track number, note density etc. Which can be done through summation of embeddings or concatenation.

2.3.2 Goal

A well known restriction of sequential models for musical composition is the length of the sequence e.g. $\sim$ 500 tokens in [26]. On larger scales, the composition looses its coherence and avoids repetition of the previous patterns, as seen in my previous work [7]. Newer model from google magenta [19] is able to generate consistent pieces through relative attention mechanism with time scale of 60s (2000 tokens). To overcome this problem and generate longer pieces, our goal is to constrict the length of the sequence and generate only small pieces that fit together and follow a guideline of a predefined musical form (which we extract from real data, but it can be prescribed arbitrarily).

For such task we need a seed measure, musical form and various sequence to sequence models, such as a generator of similar variants that elaborate on an idea of a given musical motif, a generator of next measures that fit together with the previous measure, or a generator of accompany track for some instrument and combine them together. Our goal is to train such models and generate multi-instrumental (variable of number of instruments playing up to 16) polyphonic compositions that are coherent over much larger time scales than with previous models in the field.

3. Related Work

In this chapter we describe works that influenced this thesis. They fall into three areas: deep learning for music generation, musical similarity, and works on extracting musical form.

3.1 Music Generation

Other works trying to generate music take on many different approaches and perspectives using most available representations (e.g. direct waveforms approach [27] or discrete symbolic representation [28]) and deep learning generative models (generative recurrent neural networks (RNNs) Eck et. al [29], Boulanger-Lewandowski [28], generative adversial networks (GANs) [30], variational autoencoders (VAEs) [31]).

Works that use discrete symbolic representation usually utilize either *piano-roll* (x-axis represents time, y-axis pitch) [28] or event based midi-like representation [18]. The piano-roll representation has its advantages such as being able to match a specific duration to a time step in an autoregressive model. This translates to being able to match rhythmical structure (given a time signature) by the architecture of a model e.g. skip connections over measures [7]. One potential disadvantage is the usually lower granularity as most of the related work use only the sixteenth note as its step [32] [31]. Another disadvantage is the need to model the joint distribution for the polyphonic output [28] [33]. The event-based representation [18] untangles the joint probability into a sequence. This sequence handles more effectively the silent moments and long notes while only encoding coding the time shifts and not multiple less telling values (mostly zeros).

The problem with most related works is the use of models in a way that is unable to scale to longer sequences with acceptable coherence of the outputs. The transformer model from Google Magenta [19] is able to generate consistent pieces through relative attention mechanism with time scale of 60s (2000 tokens).

Other work from Google Magenta, [31] uses the VAE generative model in a way that enhances its ability to produce longer sequences by using a hierarchical decoder. There, a "conductor" RNN decodes the latent vector into new embeddings for every measure and the corresponding measures are then generated only given the previous note, state of the note generating RNN and its corresponding embedding. However, no such mechanism is implemented for the encoder part.

Counterpoint by Convolution [34] and DeepBach [17] use convolutional neural networks (CNNs) across tracks and across a small time window both into the history and future of the composition. This model is then iteratively used by Gibbs sampling to generate missing pieces of the composition. These models are able to work with multiple tracks. A downside is the need for a fixed number of tracks due to their architecture while not allowing polyphony within the tracks. Another downside is that while the local coherence in the resulting compositions is good, these models do not ensure longer-term coherence.

Dong et al. [30] use a combination of three GANs to generate compositions with 5 tracks for bass, drums, guitar, strings and piano. It uses objective metrics for evaluating its results against the real data, such as ratio of empty measures, number of used pitch classes per bar, fragmentation ratio, drum pattern (requires known time signature) and a measure of harmony between a pair of tracks.

From works using raw audio as its data, Dieleman et al. [35] uses discrete auto encoders such as VQ-VAE model (which learns a codebook, quantises the queries to members of the learned codebook) and proposes a new model, the *argmax autoencoder* which by using softmax encourages the queries to be "close" to one-hot vectors and quantises them on a simplex to generate longer raw musical sequences over tens of seconds.

A work that used predefined (by hand) form to generate music by deep learning was a collaboration[1] based on the author's previous work [7] and a work [36]. The work of Pavlín [36] is a framework for creation of hierarchical musical forms that features repetitions and transpositions of the musical material. The musical material itself is created by randomized but sound rhythm choice specified by style and randomized pitch choices specified by key. We elaborated on this idea by adding variations to the musical form and replacing the rigid generation of the musical material by a LSTM from [7]. The variations of the musical material were simulated by "restarting" the state of the generating LSTM while having a stored state for each group of variants. The state was the last hidden LSTM state before generating first occurrence of one of the variants of a given theme (primary) for the first time. This approach allows to create more long-term coherent musical compositions as it uses an external handcrafted musical form to guide its long-term dependencies. However, such approach is not able to produce a variant given a musical material i.e., we cannot condition on what the material should look like beyond restarting the LSTM from the same hidden state. Another disadvantage is the need to handcraft the musical forms before generation.

3.2 Music Similarity

From works trying to measure how two pieces of music are similar, we are going to look at the ones that are using discrete symbolic representation as these are fit for our datasets.

According to Rizo [21], the most significant works for automated comparison of musical pieces are by Mongeau, Sankoff [2], using an adaptation of a string matching edit-distance algorithm [37], and from Selfrige-Field et. al [38], which deeply analyses the methods and representations up to 1998 date, finding that most commonly used methods are the string edit distance algorithms. However, to the date of Rizo's work [21], six trends were found in representation and comparison of musical pieces. These are string distance alignment methods, n-gram algorithms, graph encodings, statistical comparison measures, geometrical frameworks and tree representations; with each multiple exemplar works cited in [21] *p. 12.*.

A work by Stech [22] performs an in-depth analysis of melodic lines (monophonic) by detecting tonal contours, inversions, and retrograde inversions combined with matching the rhythmic structure. This work is interesting because inversion and retrograde (the retrograde was, however mostly used in serialism

[1]The work was not presented in paper, but its output was presented as a live play on Microsoft DOTS 2018 conference (video)

era) along with their combination in melody is one of the most common "more advanced" variation techniques.

Comparison of symbolic musical sequences is usually split into monophonic and polyphonic case. The latter is a much more complicated problem because of possible intercepting lines of melodies which are harder to detect by an algorithm. One of the options of how to tackle with polyphonic similarity is to try to separate these melodies by melody extraction algorithm. However, this only account for melodic similarity and also, rule based extractors struggle to extract the melodies for symbolic music, where often times the trivial skyline methods have the best performance [39].

Due to vast options of related studies, we mostly looked into the edit-distance algorithms and their polyphonic extensions [3], as they are the most commonly used. As our similarity metric stands on the grounds of the works [2] and [3], we thoroughly explain them in the Chapter 5 instead of elaborating them in the Related Work.

3.3 Musical Structure Extraction

Automatic musical structure extraction looks into detection of rhythmic information, harmony and melody lines, compositions structure and music similarities [40]. The composition structure e.g. intro, verse, chorus, instrumental or ending can be identified by self-similarity analysis and clustering.

To accomplish the structure extraction, the algorithm must be able to work with similarities within the composition to detect repetition at different time scales

From works trying to automatically extract the musical structure information, to our knowledge, most work with raw audio represented pop/rock music. Such music have distinct structural sections of repeated composition structure components (e.g. chorus), usually in a slight permutation of the following manner:

`Intro, verse 1, chorus, verse 2, chorus, chorus, outro.`

Work from Dannenberg, [41] similarly to our approach (although on raw audio data), tries to extract a musical form describing the structure of repeated phrases e.g. AABA song structure, where the letters represent a phrase. A melody based similarity approach working with chroma vectors and autocorrelation is taken to identify similar segments, which are then clustered to form patterns to form an "explanation" of the analyzed music. Dannenberg also tackles problems with finding the similar phrases, but not having any explicit indication that a phrase occurs more than once. Another problem is that the similarity metric is not transitive and that has its implications on the clustering of the phrases.

Work from Maddage [40] uses melody-based similarity regions (similar pitch contours) to detect verses and content-based similarity (similar vocal content and similar pitch contours) to detect choruses. The work is evaluated on annotated 50 English-language pop songs of 4/4 time signature, where it reaches an accuracy of 0.7.

Foote and Cooper [42] segment audio by detecting significant audio changes through the use of extracted mel-frequency cepstral coefficients (MFCCs) and constructed self-similarity matrix.

The extraction of repetition in music or extraction of the most representative part of a song (Audio Thumbnailing) was investigated by Barsch [43]. The work constructs vectors from extracted chroma-based features of segmentations of beat-synchronous frames. Then, measures similarities between these vectors to find the the chorus sections as thumbnails of the music.

Nieto [44] uses MIDI as a discrete symbolic representation for Audio thumbnailing. An advantage of this approach is a lossless reconstruction of the annotated music data. It uses 4-dimensional chromagram vectors of (pitch class $\in \{1...12\}$, energy E, time length L, polyphony N) and extracts the motives using two existing techniques for visual recognition and raw audio explained in [45].

4. Data

The data used in this work are downloaded from paid website `www.kunstderfuge.com`[1].

The reason for this dataset among other options, see work [7] (Chapter Data), is that no current classical music dataset that we know of has a wide range of instrumentation and track number ranges available. The data are handpicked to match the most important authors from late baroque, classicism and romantism eras. The data format is *MIDI*. MIDI stands for technical standard, and a data format that allows instruments, computers and other devices to connect and communicate (full specification in [46]), which means we work with discrete symbolic representation of music only.

Data contains 1537 midifiles which are organized by tree file structure into instrumentation, authors and mode of their key.

We use a term *track* to specify a part of a composition played on its own and played for its full length (given we pad it with rests), given some instrument. Tracks are played simultaneously creating polyphonic sound if some two notes are played as the same time. However, there can be polyphony within a track as well. Two or more tracks in a given composition can have same instrument assigned.

To show our design's capabilities of working with unspecified number of instruments up to 16 maximum and work with arbitrary long data (limitation for length is mostly computational cost of $O(n^2)$ for the analysis of musical form), we include around 10% of our dataset being symphonic compositions using and around 20% string quartets. The rest of the dataset (50%) is mostly written for multiple piano's, with around 20% being miscellaneous data with instruments such as harpsichords, horns, choir voices etc.

The 1537 midifiles contain 733622 non-rest measures available for training from total 1042251 measures. The roughly 300000 rest measures come mostly from symphonic compositions where many instruments are often silent.

[1]Even though the musical compositions themselves are in the public domain, their MIDI versions are contributed to the website by various contributors where they are provided with appropriate permission in a good faith.

To obtain the dataset, there are multiple ways. To avoid unintentional copyright infringement, we suggest downloading the dataset from the website, either paid or free over multiple days. We protect the attached dataset by a strong enough password which is provide to the reviewers.

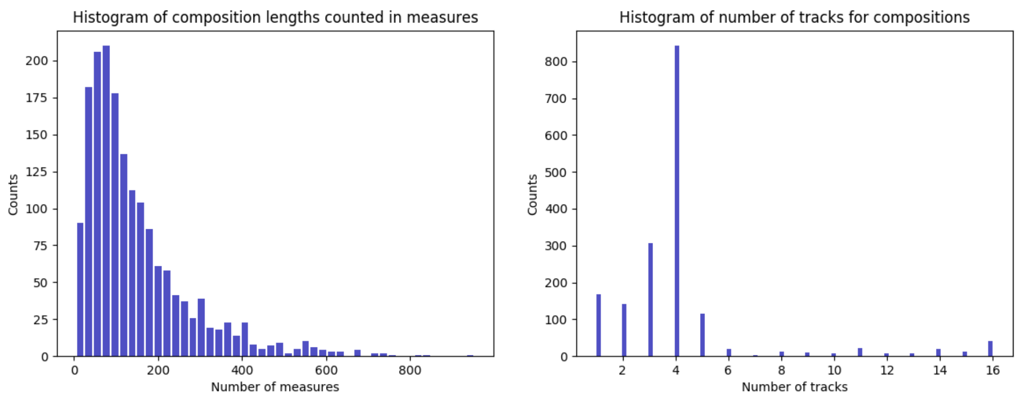


Figure 4.1: Histograms of length of compositions by measures on the left side and number of tracks on the right side

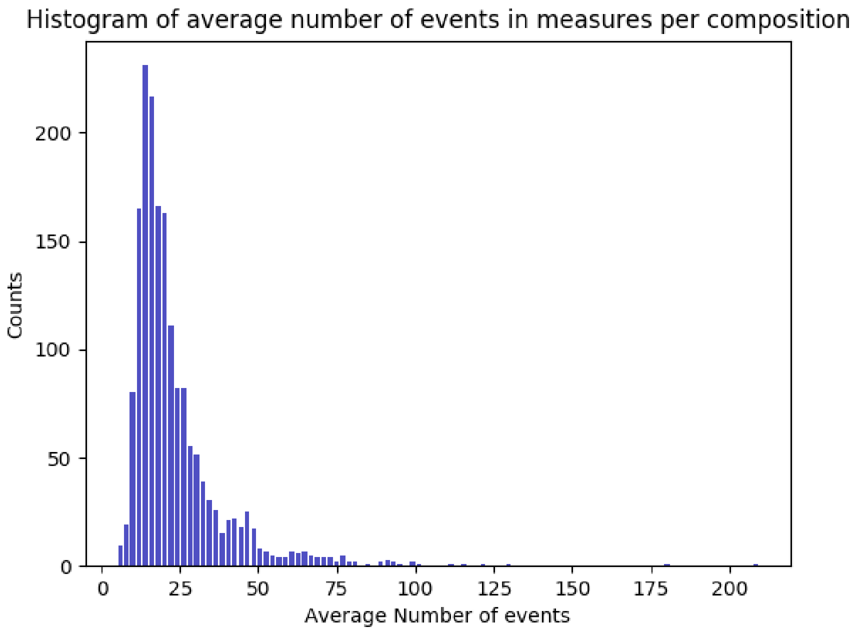


Figure 4.2: Histogram of average number of events in a measure per composition

Further technical information about the dataset is described in Attachment A.3.

We add that the MIDI files are bad at precisely timing the durations of notes. This means that to identify identical notes, we make compromise on how large of a duration difference to tolerate. We tolerate a 32th note difference for every quarter of a duration of the notes.

4.1 Data Preparation

To make use of the raw data, certain preprocessing was done. The most important is determination of the mode i.e. "major" and "minor" and "normalization" of the dataset to be of a common tonic (C) and similar average pitch (C4). The normalization to common tonic is the reason why we omit representing the key and represent only the mode. We can do such normalization, because of the use of equal temperament tuning system, described in [5] *p. 39*, in our dataset of western classical music. It allows the performances to be transposable to different pitch classes within their mode. While the key in which a composition is written might hold some additional information over just the mode (e.g. f minor as opposed to just minor) due to the tradition of certain expressive characteristics being associated with individual keys, this does not seem like an important factor at the level of sophistication this work is aiming for.

Information about the mode of a composition is necessary for the pseudometric (see Chapter 5). However, we also use it as a part of the input for the models to know which mode they should generate.

Determination of the key is done via the Krumhansl-Schmuckler key-finding algorithm [47]. However, for some compositions it is hard to decide which key they are written in. That is because the authors make violations against the mode, which help keep the listener interested and more importantly, because many compositions change their key throughout their course, in a process called modulation. However, we chose to not bother with determining the sections of keys (at a cost of less accurate pseudometric and generation) and worked with the one found by the key-finding algorithm. We worked with a parameter `tonal certainty` of the open source toolkit `music21` [48], empirically chosen to be 0.7. The compositions below this parameter were usually atonal works that we omitted (the dataset we refer to in this chapter has no "atonal" works due to preprocessing).

All of the compositions were transposed to be the most aligned to C tonic, whether major, minor or atonal. Then they were transposed to be the most aligned with $C4$, with similar method to 6.5.1 and 6.5.2.3.

The last part is mostly technical dealing with impurities within the less standardized *midi* format, described in the attached documentation A.2. That is correcting the order of note on and note off events and removing "invalid" notes from the data.

5. Similarity Model

As indicated in the System Design Chapter 2.1, one of the main requirements of the presented generation system is to generate variants for a given theme or motif. The first step towards training such a model is to have proper data for it. Obtaining such data requires processing the raw dataset composition-wise by similarity analysis, based on which we can then infer a musical form.

Similar fragments of music are usually variations of some musical material.[1] Variation is a technique which repeats a modified version of a musical material somewhere in the composition, or in a new composition. The modifications might be in melody, rhythm, tempo, harmony, timbre, orchestration etc. or combinations of these. However, the modifications are not specifically defined and as stated by Rizo [21] *p. 6*, music similarity is an ambiguous term and can be judged based on "resemblance" of these patterns (melodic, rhythmic) or harmonic coincidence.

As written by Stech [22] p.112.: *"Computers can only assist music analysts by relieving them of the burden of identifying and tabulating objectively defined musical events; they cannot be used to measure subjectively defined musical relationships.", computers can hardly find similarities that even humans are not able to define and that happens on a subconscious levels in the head of an (in)experienced listener. Therefore, we try to define the best possible objective criteria for a musical pseudometric of similarity.*

Our similarity model is based on techniques of string matching (known as edit distance [37] [49]) adapted for music [2], with slight changes in parameters according to our dataset, see 5.1.3. Further work [3] extended the pseudometric for new operations for polyphonic music. We make slight changes to the polyphonic extension making its scope less ambitious (we omit permutations between sequences and therefore, rework fragmentation and consolidation), see 5.2.2.3, some operations less strict (allowing more notes to be matched to single pitch and dividing final weight by a smaller chord), see 5.2.2.2, and solving some issues not considered in the original paper, see 5.2.1.1.

5.1 Monophonic Case

A musical composition is monophonic when at any given time there are no two pitches sounding simultaneously; e.g. a melody sung by a single singer without accompanying harmony. From musical similarity perspective, monophonic music is less complex for analysis (e.g. we have a well defined melody).

Let a melody be a sequence of notes and compare any two melodies by an algorithm that outputs *dissimilarity* (distance) between them. The two compared sequences can be of differing lengths and their (dis)similarity depends on their tonal and rhythmic structure [2]. This dissimilarity can be thought of as a distance (although we will see that it is not a *metric*) and then it could be used for further analysis of larger number of melodies. Such analysis could be hierarchical clustering to see how classes of melodies can be divided into subclasses or projection into two dimensional space to visualize variability among melodies [2].

[1]similar to a degree that is discussed in the Chapter 6

The pseudometric quantifies the *dissimilarity* between two musical sequences A, B by minimal weighted transformations applied to the first sequence to get the second one. Such transformations are *insertion, deletion, replacement, consolidation* and *fragmentation*. The sequences consist of notes that have certain pitch and length, or of rests which only have their length. *Trace* of such transformations is a sequence of lines illustrating replacements, identities, fragmentations and consolidations between the two sequences of notes.

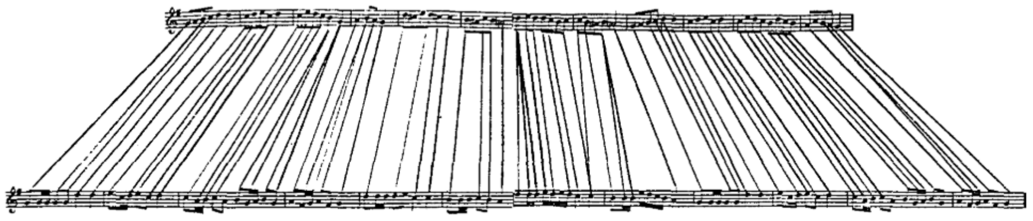


Figure 5.1: Illustration of traces between two monophonic sequences from [2] *p. 169 "Figure 5. Trace linking Variation 2 to Variation 3."*

5.1.1 Data Representation

The musical sequences are encoded as ordered series of tuples, each representing a note in time:

$$(P, R, L, S, D, O) \tag{5.1}$$

where P is a pitch

R is an indicator of rest

L is length of the note

S is a scale indicator (more specifically, mode of a key 1.2)

D is degree in the scale according to the key

O is an indicator for a note that is out of the set of pitches defined by the key

The length of notes is measured not in time but in beats, or rather in sub-beat units derived from a shortest viable" note, which we choose to be a 32nd note (so an 8th note has a length of 4). This makes the representation tempo-invariant. If R indicates a rest, all other variables except L are ignored.

Ideally, we would like to have the data invariant to key, meaning we forget the pitch and analyze the notes by their distance from the tonic of their key. However, not all data have their key easily detectable. We analyze the key by Krumhansl-Schmuckler key-finding algorithm [47], but for some compositions, it is hard to decide in which key they are, either because they might be atonal or there might be switching between the keys throughout the composition and we are not sure what size of a window to use for sub-analysis down to a measure levels. We then end up in classifying some compositions in scale *"atonal"* and just remember the distances in semitones from its tonic. Elegant way to get these distances is by transposing our whole dataset into a common tonic and then work with their pitches as pitch classes [2] while disregarding their octave, as the octave does not alter the tonal contour of melody.

5.1.2 Transformations

To quantify the minimal number of transformations to be applied to sequence $A = (a_1, a_2, ..., a_n)$ to obtain a sequence $B = (b_1, b_2, ..., b_m)$, we use typical[3] transformations like *insertion*, *deletion* or *substitution* of a note in a specific position along with two new operations, *fragmentation* of a note into several shorter notes and *consolidation* of several short notes into one longer note.

A generalization of this concept is weighting each transformation [49]. The weights for each transformation are determined by differences of length of the note(s) involved and by the differences in their pitches according to rules of consonance.

Insertion with weight $w(\varnothing, b_j)$ is insertion of b_j into B, *deletion* with weight $w(a_i, \varnothing)$ is deletion of a_i from sequence A. *Replacement* with weight $w(a_i, b_j)$ is substitution of a_i by b_j, where $a_i = b_j \implies w(a_i, b_j) = 0$.

Consolidation with weight $w(a_{i-k+1}, ..., a_i, b_j)$ is replacement of several notes $a_{i-k+1}, ..., a_i$ by a single one b_j, where *fragmentation* $w(a_i, b_{j-k+1}, ..., b_j)$ replaces one note by several. These two transformations are motivated by cases, where we replace e.g. *whole* note with four *quarter* notes (i.e. the same total duration) of the same pitch which is perceived musically similar, but it would be very costly with only the three basic transformations (multiple insertions and deletions).

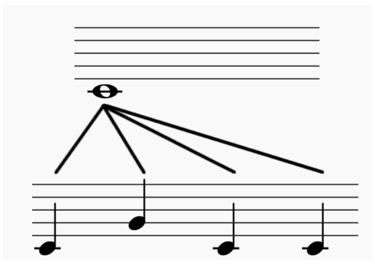


Figure 5.2: The fragmentation of a longer note into shorter notes of similar tonal contour and the total size similar (or the same) as the first note.)

Therefore, the weights for these two transformations are determined by comparing the the pitch classes between each of the replacing notes and the replaced note and the total length of the replacing notes with the length of the replaced one.

The dissimilarity between $A = (a_1, a_2, ..., a_n)$ and $B = (b_1, b_2, ..., b_m)$ is determined recurrently through the minimum of sum of all weights among all possible series of transformations that convert A to B. The dissimilarity d_{ij} recurrence

[2]pitch class is distance from the tonic counted in semitones
[3]typical in context of edit-distance[37]

equation is

$$d_{ij} = \min \begin{cases} d_{i-1,j} + w(a_i, \varnothing) & \text{(deletion)} \\ d_{i,j-1} + w(\varnothing, b_j) & \text{(insertion)} \\ \{d_{i-1,j-k} + w(a_i, b_{j-k+1}, ..., b_j), 2 \leq k \leq j\} & \text{(fragmentation)} \\ \{d_{i-k,j-1} + w(a_{i-k+1}, ..., a_i, b_j), 2 \leq k \leq i\} & \text{(consolidation)} \\ d_{i-1,j-1} + w(a_i, b_j) & \text{(replacement)} \end{cases} \tag{5.2}$$

for $1 \leq i \leq n$ and $1 \leq j \leq m$, where initial conditions are

$$\begin{aligned} d_{i0} &= d_{i-1,0} + w(a_i, \varnothing), i \geq 1 && \text{(deletion)} \\ d_{0j} &= d_{0,j-1} + w(\varnothing, b_j), j \geq 1 && \text{(insertion)} \\ d_{00} &= 0 \end{aligned} \tag{5.3}$$

The final dissimilarity value is then $d_{n,m}$, and the traces matching the notes that were part of identity, replacement, fragmentation or consolidation can be obtained by using pointers from d_{nm} to each d_{ij} of the previous recurrence equation.

The weights are in general defined as:

$$w(a_i, b_j) = w_{interval}(a_i, b_j) + k_l \cdot w_{length}(a_i, b_j) \tag{5.4}$$

where $w_{interval}(a_i, b_j)$ is predefined weight based on the relative interval between the two notes multiplied by the length of the shorter of the two notes, $w_{length}(a_i, b_j)$ is the difference between the durations of the two notes and k_l is a constant that represents the weight of contribution of duration differences against contribution of tonal differences. As for the weights for *insertion* and *deletion*, the $w_{interval} = 0$ because the length of the shorter note, i.e. empty note, is zero. Weights for *fragmentation* and *consolidation* are also similar to the general definition. For consonance part, there is summation of all the corresponding $w_{interval}$ between the fragmented (consolidated) note and its several counterparts, and for length difference part, there is the duration of the fragmented (consolidated) note against the summed duration of all its counterparts. Rests cannot be fragmented or be consolidated from notes because it severely disrupts the rhythmical similarity. As we will see in next paragraph, $w_{iterval}$ for rests is fairly low regardless of compared the pitch and the total length of **any** rhythm can match exactly the length of rest.

Time complexity of this algorithm remains quadratic on the condition we change the upper bound of k in *fragmentation* and *consolidation* part of equation 5.2 to be bounded by constants determined by length ratios between the involved notes and the fragmented/consolidated note [2]. The logic behind these constants is that for *fragmentation* or *consolidation*, we can always find constant, after which the cheaper transformation is to perform numerous *insertion*/*deletion* operations than to extend the scope of the former two operations (e.g. we know the total length difference weight component between the fragmented note and a certain number of notes will only be larger if we add more notes into the scope).

5.1.3 Parameters

In the equation 5.4, there are parameters for determining of $w_{interval}$ and a parameter k_l which is dependent on the dataset and is determined by observations. Such observations could be alignment of two highly dissonant notes of same length pushing the value of the parameter down, against fragmentation to a set of notes whose total duration is very different than the duration of the fragmented note pulling the parameter up. We used a value slightly larger than by authors [2], $k_l = 0.460$, as our requirements were for more rhythmically similar fragments.

The parameters defining the interval weights $w_{interval}(a_i, b_j)$ are designed to match are consonance of pitches and are determined modulo octave since compound intervals 1.3 are on a similar level of consonance, so we technically work with only 12 semitones. Different set of weights are used whether the considered notes are part of the scale of their respective modes, Table 5.1, or at least one of the notes is out of the scale, Table 5.2. If both notes are *rests* then the interval weight is 0. If one of the notes is a *rest*, interval weight is 0.4, which is higher than this parameter from the authors of [2] (0.1), because we want to ensure the rhythmical stability and matching too many notes to silence disrupts that.

As the authors of [2] describe: "We can be really confident only about the order of weights in terms of decreasing consonance. The precise values we have calculated are debatable, or could be optimized with respect to some data set.". In this thesis, we chose the same values for degree differences as in [2] and slightly alter the values downwards for differences one or both of the compared notes are out of the set of pitches determined by scale:

Degree differences

difference	0	1	2	3	4	5	6	rest
weight	0	0.9	0.2	0.5	0.1	0.35	0.8	0.4

Table 5.1: The weights used when both of the compared notes are in their respective scales and we view them as degrees of it, rest column specifies the weight if one of the notes is a rest

Out of scale differences

difference	0	1	2	3	4	5	6	7	8	9	10	11	rest
weight	0	2.4	2.1	0.6	0.6	1.2	1.4	0.4	1	1	2.0	2.3	0.4

Table 5.2: The weights used when at least of the compared notes is not in its respective scale, or the composition scale was not properly detected, rest column specifies the weight if one of the notes is a rest

However, the final (dis)similarity only a pseudometric. From the cognitive perspective [50] a similarity measure is not transitive. Adding two consonant intervals can yield a dissonant interval, therefore, it cannot satisfy the triangular inequality. Nevertheless, our handling of the results takes that into account.

5.2 Polyphonic Case

Polyphonic case poses a more difficult problem. Some systems try to reduce polyphony to monophony and use the appropriate methods on the reduction, but that comes with the problems of its own for the reduction algorithm and a loss of information.

We will be working directly with polyphonic data. The final polyphonic algorithm is an extension of the monophonic algorithm described in section 5.1.3 and is a slightly modified version of the algorithm proposed by Ferraro et al. [3].

Representation options for polyphonic data are much broader as there are multiple notes playing at the same time. We could also determine what comprises a melody, what are the accompanying chords forming the harmony etc.

Transformations between chords are harder to determine, as there are more options for matching the intervals between all the notes involved. For our purposes, chords are unordered sets of notes and the transformations should find the best possible matching of the two sets of notes. Polyphony in a given time may even match several consecutive notes regardless of their permutation or repetition of some of them e.g. *broken chords (arpeggio).*

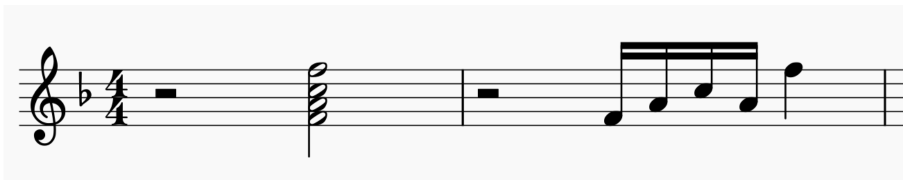


Figure 5.3: The two successive measures are similar as the first chord is arpeggiated, or broken into several consecutive notes. The most usual arpeggios are successive uprising or downfalling notes of the chord, but there can also be other permutations or repetitions

5.2.1 Data Representation

The main difference against monophonic representation in representing the polyphonic data is to have a data structure, that allows several notes sounding at the same time. Representing chords as unordered sets of notes is a well established representation [51]. Chords are required to be unordered because of arbitrary order of pitches for note-to-note comparison can estimate perceptually similar chords as dissimilar [3]. For unordered chords, we have to find the best permutation of both of the sets of notes involved in a transformation.

We change the representation in monophonic case 5.1 so that P, D and O are now acting as sets of pitches, sets of their respective degrees and sets of their respective indicators of being out of a given scale respectively.

5.2.1.1 Passing Notes

The selected representation encounters a representation problem, where there is unclarity on how to represent a situation when a chord a is transitioning from a chord (or single note) b constituted by a subset of notes from the chord a, while the transition does not articulate the mutual notes. The same goes for the other direction i.e. a chord transitioning in the same manner into a unarticulated subset of its own notes.

The straightforward solution of making the unarticulated transitions viable for our data representation while keeping their information value is to get a use of *fragmentation* and *consolidation* transformations and simply articulate these transitions.

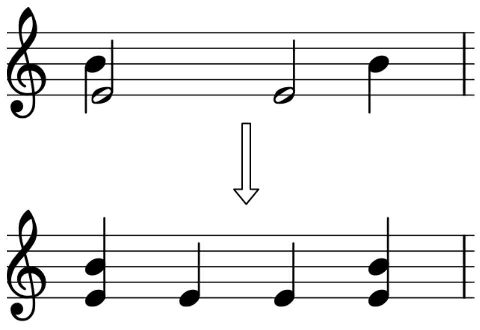


Figure 5.4: Fragmentation of passing polyphony.

5.2.2 Extended Transformations

We will use all the general transformations defined in the previous monophonic section, but the transformations will be extended for working with chords. The extensions will be defined more generally and they will not differentiate between single notes and chords, while keeping the the functionality for monophonic sequences.

Extending the defined monophonic transformations 5.1.3 for chords is not as straightforward as setting new scoring weights. Even when working in only one octave and assuming a reasonable maximum size of chords[4], there is a combinatorial explosion [3] *p.8*.

Instead, we keep the parameters of $w_{interval}$ and try to find the minimal matching between the sets of notes while allowing some of the notes in the chords to be added or deleted.

5.2.2.1 Insertion and Deletion

Extending operations of *insertion* and *deletion* is simply performing the operation for each note in a chord independently. However, these two operations will be now used as a part of operations *replacement*, *fragmentation* and *consolidation*.

5.2.2.2 Replacement

Our representation of chords C as unordered sets of notes leaves us with a task of finding the best permutation of notes in chords involved in a transformation [52]. In other words, we seek a matching between the two sets of notes while minimizing the sum of weights in matching.

We also want to allow the notes to be left out of the matching if the weight of the matched replacement exceed the weight of their insertion or deletion.

[4]In practice, chords rarely exceed the size of 4

This problem can be modeled as *assignment problem* [53] and can be solved by any version of Kuhn–Munkres algorithm in polynomial time [54].

Weight of replacement between two chords (or single notes) $w_{poly}(C_1, C_2)$, with chord sizes[5] k and $l \in \{1, ..., 4\}$ is is calculated using the solution to a minimal perfect matching on a complete bipartite graph $G = (U, V, E)$, where:

> $U = \{x_1, ..., x_k, \lambda_1, ..., \lambda_l\}$ is a vertex set for the first partition, $x_1, ..., x_k$ represent notes in the chord C_1 and $\lambda_1, ..., \lambda_l$ represent $\varnothing$ as empty note for each of the notes in the second chord C_2
>
> $V = \{y_1, ..., y_l, \epsilon_1, ..., \epsilon_k\}$ is a vertex set for the second partition, $y_1, ..., y_l$ represent notes in the chord C_2 and $\epsilon_1, ..., \epsilon_k$ represent $\varnothing$ as empty note for each of the notes in the first chord C_1
>
> Weights for the complete bipartite graph are set as:
>
> > $w(x_i, y_j)$, for edges of type $e = (x_i, y_j)$ for $i \in \{1, ..., k\}$ and $j \in \{1, ..., l\}$
> >
> > $w(x_i, \varnothing)$ or $w(\varnothing, y_j)$, for edges of type $e = (x_i, \epsilon_{\bar{i}})$ or $e = (\lambda_{\bar{j}}, y_j)$, for $i, \bar{i} \in \{1, ..., k\}$; $j, \bar{j} \in \{1, ..., l\}$
> >
> > 0, for edges of type $e = (\lambda_j, \epsilon_i)$ for $i \in \{1, ..., k\}$ and $j \in \{1, ..., l\}$

In addition to what the authors write in the [3], we do two slight following changes. To counter the weight disadvantage of larger chords against smaller chords or single notes, that comes from summation of matched intervals in the perfect matching we divide the minimal perfect matching by size of the smaller chord (or note i.e. size of 1).

For replacements involving *rests* it comes from the the general extended definition as $w(\mathit{rest}) \cdot \mathit{size}(C)$ multiplied by the length of a shorter note involved in the replacement.

The second change is that for notes that were added or deleted, i.e. the edge involving a variable representing $\varnothing$ was selected in the perfect matching solution, we allow the definition of a matching to be violated by a possibility of matching more notes to one. This aims to solve the cases of uneven sizes of chords, where e.g. a chord of one note[6] can have reasonably small replacement weight against all the notes in the second chord, but is forced to add or delete the notes in the second chord. Therefore, for added or deleted notes, we try to find a possible lower match.

5.2.2.3 Fragmentation and Consolidation

These transformations are defined similarly to monophonic case, but their extensions play a key role in finding similarities between chords and consecutive notes/chords i.e. broken chords/arpeggios 5.3. Reasonable sizes of chords and local boundaries for number of consecutive notes set in the same fashion as in monophonic case does not increase the asymptotic time complexity.

[5]Chord size is counted in number of different pitches in a chord, not to confuse with chord duration

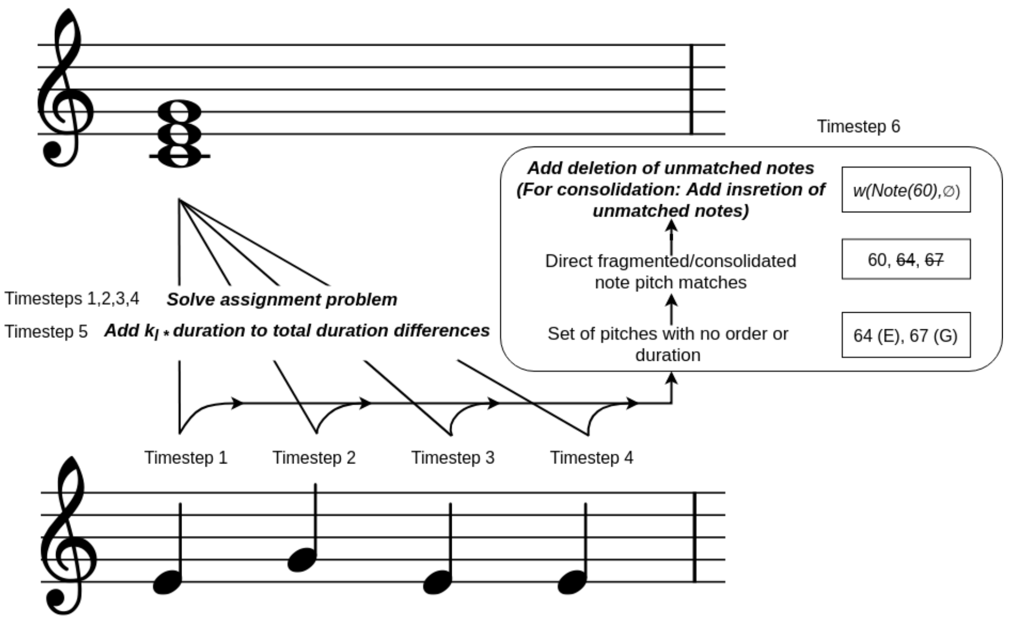
[6]Technically not a chord, but we can view it as it is by our definition of the transformations

In [3], these two operations consider even permutations in the order of notes for sequence to sequence operations (due to their representation). We are less ambitious and rework these two operations, but keep all the other ideas from the [3] work, i.e. we keep finding any permutations, repetitions within arpeggios.

In addition to extending *fragmentation* and *consolidation* using $w_{poly}(C_1, C_2)$ instead of the monophonic weights, we'll save the consecutive(fragmented or to be consolidated) chords into memory using undordered set representation.

The following explanation will be made on *fragmentation* transformation as the *consolidation* can be thought of as an inverse operation. For *fragmentation*, we retain all the consecutive notes/chords $C_{2,i}, C_{2,i+1}, ...C_{2,j}$[7] that are part of the transformation $w_{poly}(C_1, C_{2,i}, C_{2,i+1}, ...C_{2,j})$ for later use. To cover the cases of repeated or permuted notes/chords of differing durations, we represent them as unordered set of pitches without duration, i.e. the sequence of chords is represented by a single unordered set P.

The final weight associated with *fragmentation* is then modified by adding the weights for *deletion* of notes in C_1, for which the pitch is not found in the retained set P. Similarly for the *consolidation*.



Timestep 6
Add deletion of unmatched notes (For consolidation: Add insretion of unmatched notes)
w(Note(60),∅)
Direct fragmented/consolidated note pitch matches
60, ~~64~~, ~~67~~
Set of pitches with no order or duration
64 (E), 67 (G)
Timesteps 1,2,3,4
Solve assignment problem
Timestep 5
Add k_l * duration to total duration differences
Timestep 1
Timestep 2
Timestep 3
Timestep 4

Figure 5.5: Fragmentation (consolidation) operation on two polyphonic sequences favoring arpeggio matches as more similar than the standard (polyphonic versions of) fragmentation and consolidation. (The second sequence in this Figure is monophonic for better illustration purpose.)

5.3 Discussion and Future Improvements

Outside of many other approaches discussed in the Related Work 3.2. The approach using techniques of string matching could be further improved. For instance, in [3], there is a notion of local rearrangements of notes, which is similar

[7]In a more strict notation that makes the text harder to read: j denotes a position of chord $C_{2,j}$ in a sequence $B = (C_{2,1}, ..., C_{2,m})$, $i = j - k + 1, 2 \leq k \leq j$, ($i$, that lies in the main text)

to how polyphonic *fragmentation* and *consolidation* matches several consecutive notes regardless of their permutation or repetition in a local scale.

A disadvantage of the system that uses degrees in a musical key is that estimating the key is not a straightforward process if the key changes throughout the composition.

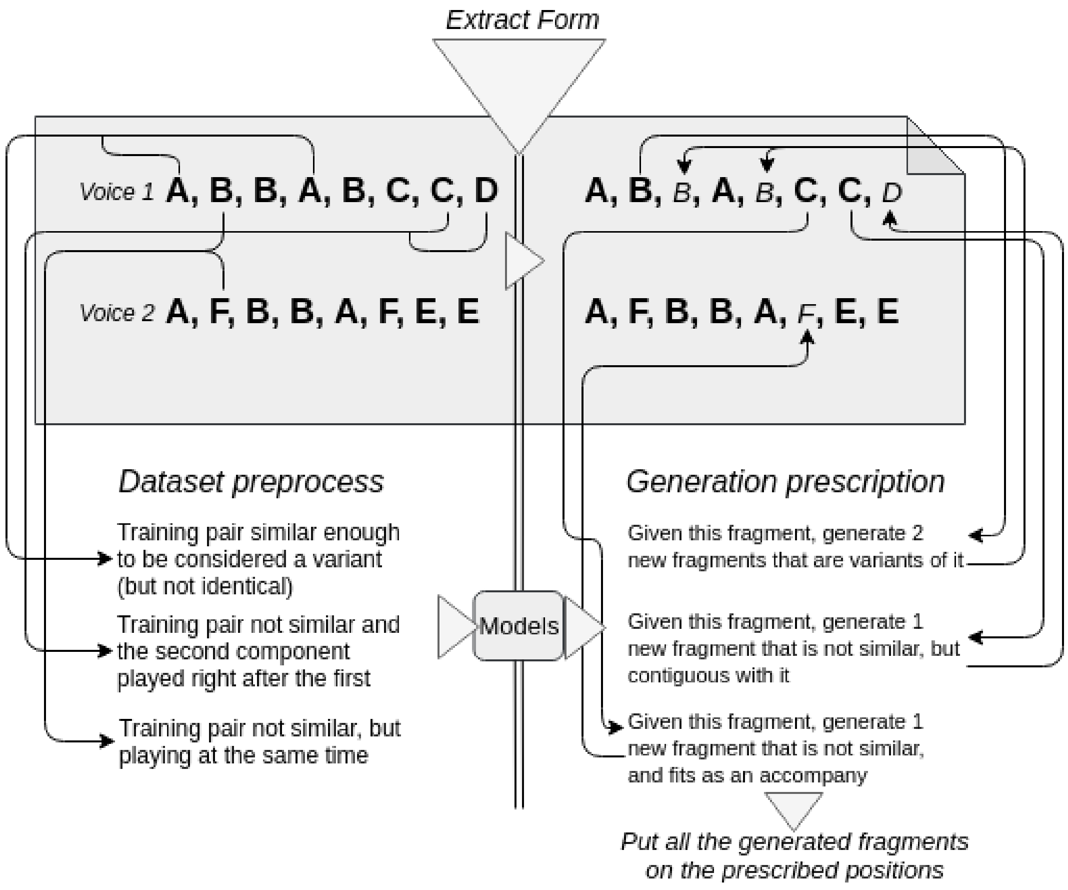
The following improvements were investigated, but without solid experiments conducted: One idea is to look for techniques which are used for variation of some themes. One can look for signs of melodic inversion or retrograde in a local scale, or a harmonic inversion of chords which are all common variation techniques. Another interesting addition to the string matching approach is multiplying the note weights by a coefficient corresponding to a relation of beats of both notes involved, i.e. notes both situated in a *downbeat* could be slightly more penalized for their possible dissonance because the listener is usually paying more attention to these beats.

6. Musical Form Extraction

In this chapter, we use the pseudometric defined in Chapter 5 for an analysis of the structure of musical compositions, which will then be utilized for training the music generation models. In essence, musical form describes which parts of a composition are similar to each other, and which are dissimilar. This balance between the surprising and the familiar is a major factor that makes a piece of music interesting. Our approach to generating music will try to imitate this principle.

First, we define the desired output, a musical form, then, we explain how the pseudometric is used to determine similarities for fragments in the compositions, and last, we look at how we cluster the raw (dis)similarity data into a sequence of terminals which constitute the musical form and indicate what parts of a composition are probably variants of each other.

The resulting musical form not only specifies the training fragments of compositions for the models that generate variants for a given theme and non-variants for a given theme, but also works as a blueprint for generating new compositions because it shows how to concatenate generated fragments into a meaningful piece.



Extract Form
Voice 1 A, B, B, A, B, C, C, D
Voice 2 A, F, B, B, A, F, E, E
A, B, B, A, B, C, C, D
A, F, B, B, A, F, E, E
Dataset preprocess
Training pair similar enough to be considered a variant (but not identical)
Training pair not similar and the second component played right after the first
Training pair not similar, but playing at the same time
Models
Generation prescription
Given this fragment, generate 2 new fragments that are variants of it
Given this fragment, generate 1 new fragment that is not similar, but contiguous with it
Given this fragment, generate 1 new fragment that is not similar, and fits as an accompany
Put all the generated fragments on the prescribed positions

Figure 6.1: Extraction of a musical form provides both the straightforward way to get the meaningful training data and a the way to properly call the models when needed along with prescription for concatenating the generated fragments. In the top-right part of the picture, un-bold letters in italic signify the fragments that have no realization (either seed or generated in previous time step) yet.

6.1 Definition

Musical form defines the structure of a musical composition by various interconnected and overlaying features of musical materials, e.g. the arrangements of instruments, arrangements of rhythm, melody, harmony, dynamics and most notably, the more abstract requirement of balancing of repetition and variation of these features [13]. For an analysis, there is a requirement for a formalization. However, for such complex sets of features and because of each composer's different imagination, there remains vagueness in the scope of description, and in the levels of temporal hierarchy. Therefore, we look mainly into the two main requirements [14] p.23, i.e. some differentiation of musical materials and relation of these differentiated materials by repetition, along with clearer definitions of features, such as choice of instruments, dynamics, density of notes, orchestration and motion.

To describe a musical form, musicians use a simple system of labeling units of musical material with letters. The first musical material is labeled A. Other contrasting materials are labeled B, C etc. [55]. If any of the material is repeated in a varied form, the corresponding variation is labeled by a number, e.g. $A1, A2, B2, ...$. Subdivisions of larger musical material are labeled by lowercase letters. Slight variations are labeled with single apostrophe marks [55].

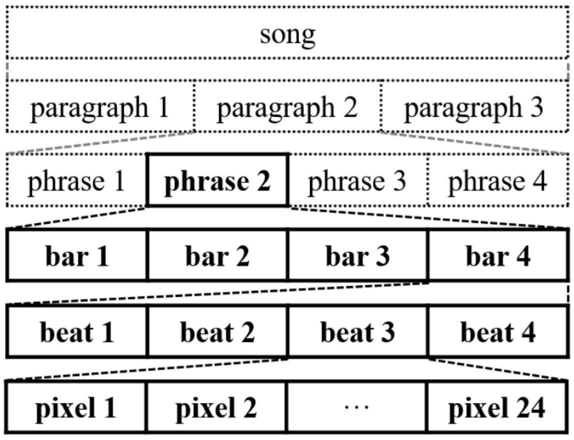


Figure 6.2: Illustration of the hierarchical structure in music from [30] at *p. 1*

6.2 Similarity Analysis

In this section, we explain how we analyze the compositions to get pairwise (dis)similarity values between the yet unspecified parts of a composition. These values are later used for clustering of the similar musical material into clusters of variants of each other. The clusters represent the musical material that comprises similar parts in the composition (i.e. $A1, A2, A3...$). Item pairs from the resulting clusters are then used to train the model to produce a musical fragment similar to another musical fragment.

6.2.1 Analysis in the Time Domain

We considered two main ways of how to approach the similarity analysis of compositions regarding alignments and granularity of its analyzed subsections. The more robust approach of finding the best alignment (i.e. ordered best similarity start-to-end bounds for the two subsections [2]) of similarity between musical sub-materials, and the more clearly defined approach of finding similarities over a set of musical fragments of predefined length. The former approach allows identifying the possibly overlaying boundaries of fragments of differing lengths along with their (dis)similarity value. The former approach also allows for different scopes of similarity so it can find the hierarchical structure of the composition. The latter approach of arbitrarily predefining the sets of fragments to be analyzed by similarity pseudometric leaves us with problems of not capturing the true similarity of musical materials that are not aligned with the bounds of the predefined fragments. As D. Stech [22] states: *"The most significant limitation observed in previous studies had to do with the arbitrary establishment of one fixed length of musical pattern for an entire analysis. Such a procedure was far too limited to take into account the many different ways in which melodies could be aurally interpreted. The complexity of the pattern analysis problem requires* *that any procedure used for melodic analysis be capable of detecting patterns of many different lengths, regardless of their interaction with other patterns, or their location within the measure."*

The pseudometric defined in Chapter 5 can be easily made to find the best similarity alignments within a whole composition [2] Sec. 9 p. 171. The outputs are the boundaries of subfragments constituting similar pairs, along with their (dis)similarity, giving us the first n best alignments. However, apart from a problem of determing n, the computational costs of this algorithm is n times larger than the cost of determining the similarities for arbitrarily predefined fragments and proved to be critical for our implementation.

We work with the limitations of the approach of fixed lengths of fragments, which disallows us to capture musical motives and phrases that are not aligned with our choice of fragmentation. By the nature of the pseudometric coming from string matching techniques, small misalignments do not pose a big problem, as these translate into a few small weighted *additions*, *deletions* or a slightly more costly *replacements*.

Furthermore, in author's experience, the units of musical material are usually aligned to some number of measures. As our generation models are stochastic and are to be fed with these fragments, we can expect them to capture the "most usual" alignments into units of measures. Therefore, our chosen "arbitrary" defined fragments correspond to the measures in analyzed compositions.

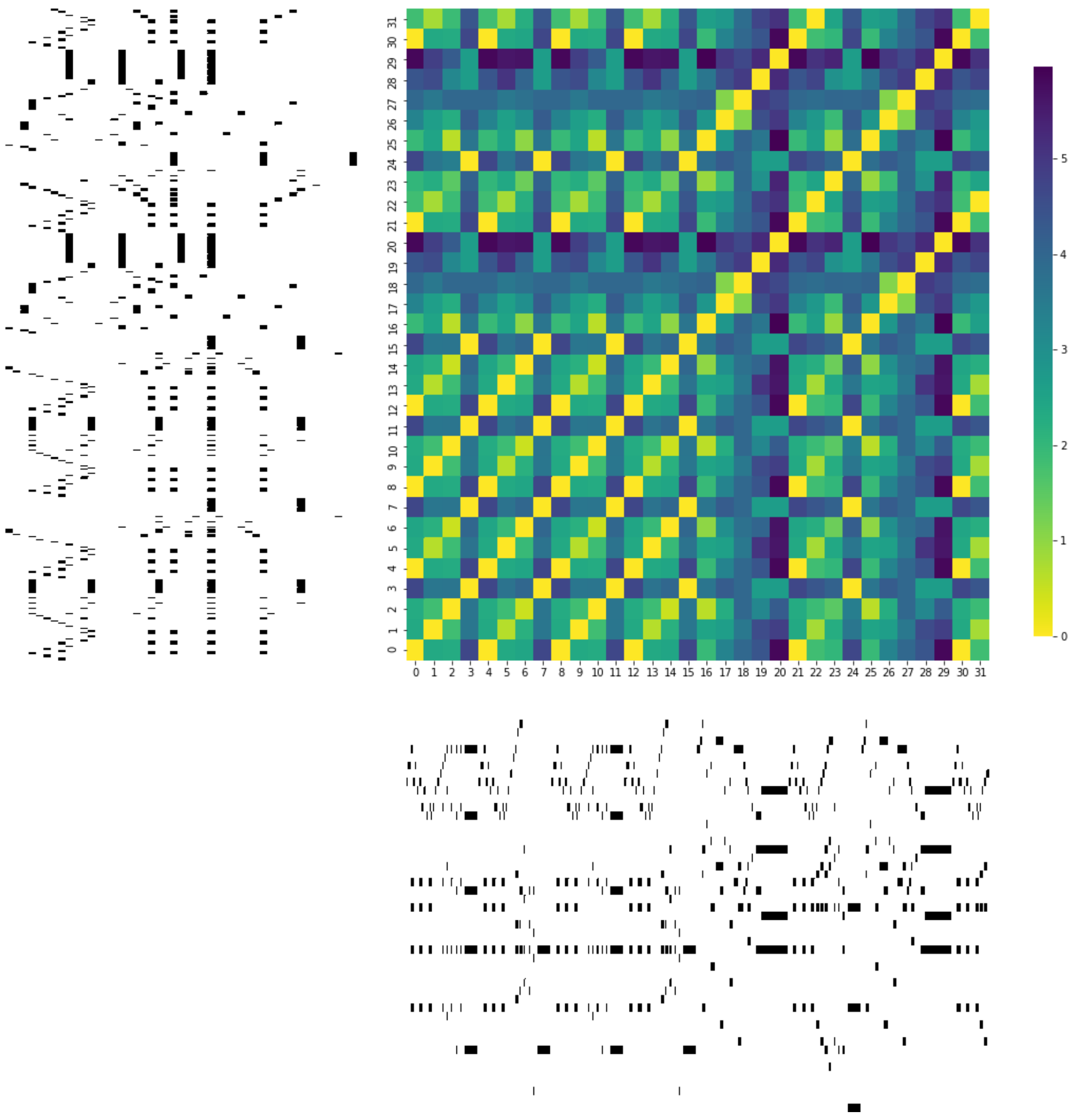


Figure 6.3: Similarity matrix of the *fixed lengths approach* for the length of measures. The figure shows first 32 measures of Beethoven, String Quartet No. 4 in C minor, opus 18 no. 4, 4. Allegro.

The *best alignments approach's* property of implicit hierarchical structuring of fragments is, to some degree, estimated by the clustering of the similarity analysis outputs of the *fixed lengths of fragments approach*. That is, phrases and figures could correspond[1] to measures (our arbitrary fragments length), while larger materials, such as motifs, are made up of these measures. Their similarity against other motifs is implicitly estimated by number of same fragments and variant fragments. All in all, the meaningful material should be directly, or in consequence of the the previous argument, indirectly recognized.

Nevertheless, the main unresolved problem of the chosen approach lies in the inability to find similarities between motives of differing overall duration in any of the structural hierarchy level.

[1]They are defined by meaning, not by the length. In cases when they are formed by several measures, we can replace them by an arbitrary unit, that we, in general want to be aligned to measures (as stated in previous paragraph, unaligned material is left out of the consideration).

From the viewpoint of utilizing the extracted musical form by the generating models, our chosen *fixed length fragments* approach has another disadvantage. That is, for musical material spanning over several units of measures, the models that are fed with fixed length fragments cannot properly analyze large scale motions and other long term dependencies, apart from those imposed by smartly conditioning the models. Example of this problem could be the possibly differing results from generating a variation of fragment of several measures against generating independent variations of its subfragments, the measures, and concatenating them together. The long term dependencies are, however, imposed on the composition by using the extracted form as a prescription in the generation process.

6.2.1.1 Note on Inter-opus Analysis

An intercomposition analysis, more specifically an analysis of individual fragments between compositions , is another concept worth consideration. Although it holds little value for the goal of extracting a musical form[2], it can provide more data for the generation models that generate variants given a theme. However, the quadratic growth of the similarity comparisons deems this task not worth our effort.

Furthermore, even with intracomposition analysis, variants that are identical in between compositions are connecting the clusters. Nevertheless, these cannot merge the clusters in terms of providing pairs with both components exclusive to their clusters, as the intercomposition analysis would, i.e. these do not widen the training examples.

On the other hand, while not providing the exclusive examples as pairs of training data, they can be in the generated fragments as a consequence of training. For example, we want to generate multiple variants of one theme on trained variants mode in sequentially as the arrow indicates, and the model generates fragments resembling: Cluster $C_1 \longleftarrow$ shared example S; $S \longleftarrow C_2$. The underlying problem here is that we have no guarantee about the worst case dissimilarity while interconnecting themes across compositions. However, apart from omitting identical fragments from training (not desirable) or designing a more complicated system, there is no way around this possible disadvantage.

6.2.2 Dealing with Multiple Instruments

In our context, we can have two concepts of polyphony as most compositions are written for multiple instruments or performers. Our first meaning, is the inherent polyphony within *tracks* [3] of the composition. Our second meaning is polyphony made by playing the tracks at the same time.

We are going to analyze the individual polyphonic *tracks* by the pseudometric 5. The polyphonic dependencies of our latter concept of polyphony(individual *track* instruments playing simultaneously) are omitted in the similarity analysis. The overall similarity perception of compositions as wholes is estimated by the individual *track* fragments similarity groupings in the extracted musical form i.e.

[2]Possible value is for determining “bounds” for fragments that hold “some” meaning

[3]Track is meant to be played by one player, or in some context by one hand of a player

we extract information about the fragments that form the B meaning polyphony. Indeed, similar melodies may be perceptually dissimilar when accompanied by different harmony or by silence. We will know that the harmony is different, however, the overall (dis)similarity remains unknown.

To further help with the estimation of overall similarity and to provide instrument dependent variation data, all the individual instruments fragments are analyzed between themselves, interinstrumentally[4]. Benefit of interinstrumental approach is that the *tracks* for instruments in compositions are usually split by some meaning and analysing them without distinction potentially loses information, just as in the opposite case, when we lose polyphonic dependencies information. Therefore, the final extracted form not only identifies arrangements of similar and dissimilar material, but also the number of *tracks* in a composition, the instrumentation and similarities within the same time frame.

6.3 Our Definition of Musical Form

In this section, we define, in a more clear way, the representation of the musical form for our purposes, i.e. 7.1. We also define the additional information that we add to the general representation of a form definition.

The musical form, in our case, holds two meanings for which the definition is adapted. The first is to provide pairs of similar and dissimilar fragments for submodels of our generation model. The second is to provide a prescription of how to arrange the outputs of the submodels together to form a composition.

The form defines not only arrangements of similar and dissimilar fragments in time, but also similarities within the interinstrumental polyphony and also functions as a "conductor" by telling each voice when to play and when not.

It is a grid of $n \cdot k$ tuples we call *terminals*, n is the number of measures and k is the number of tracks. Each measure-long time step of n total time steps is represented by k such subsequent tuples. The tuples are of form

$$(P, V, T, I, N) \tag{6.1}$$

P (for *primary* represents a musical material that the fragment belongs to. To represent values of P, we use capital letters $A, B, ...$ Every instance of such musical material is in some sense similar or even identical. Fragment pairs with the same value of P, which are not identical, are used as training examples for the generation model for variants. The special "dummy" primary marks instances of musical the material, which are not similar enough to any other fragment, including other "dummy fragments". Another special primary, the "_", marks empty fragments, which hold only rest(s) and not notes.

$V \in N$, called *variant*, in conjunction with P enables us to distinguish if the instance of a musical material is truly identical, or just similar. For example, in form PV, fragments $A1$ and $A1$ are identical and $A1$ and $A3$ are similar. The degree of similarity between different *variants* is not defined.

[4]Meaning we can get variants of fragments that are sounded at the same time, on different instruments

$T \in Z$ stands for transposition and indicates estimation of chromatic transposition (see 1.3) in the number of semitones and direction (positive for upward transposition, negative for downward) against a fixed pitch. When subtracted for two individual fragments, this is either literal transposition, in case the Ps and Vs are equal for both fragments, or an estimated transposition when the Vs are not equal.

I is a numeric midi key to a dictionary of predefined instruments or instrument classes, see A.4.

$N \in \{0, ..., m\}, m \in \{0, ..., 15\}$ is the index of a *track*, given the ordering of tracks in the data, in which the fragments lies. In the arrangements of terminals, N is practically an index of the terminal, starting from 0, modulo $m + 1$. It could be replaced by the number m accompanying the definition of form as a sequence of tuples.

As stated previously, the degree of similarity between different *variants* of a *primary* is not defined. Moreover, the degree of dissimilarity between different *primaries* in a composition is also not defined. However, later in this chapter, see 6.4, we'll see that the degree of (dis)similarity between the fragments (not) constituting a *primary* in a composition is directly influenced by the number of the "yet found" variants in them (as it is a continuous process).

6.4 Detecting Trivial Relations

To obtain the form as defined in the previous section, we first find the most trivial similarities, that can be found without the use of the more computationally expensive pseudometric. And without presenting these trivial pairs to the generating models as training data. The use of the pseudo metric on trivial cases could even provide ambiguous results (e.g. minimal cost of consolidation equals identity replacement).

The most trivial relations, or transformations, are identities, or identities with some relative chromatic transposition of the whole fragment. These are important to be clearly captured by the form, as they play major role in how we perceive musical pieces 1.6. They are also important, to not be viewed as training pairs of similar examples for our variant generating models, because such transformations holds no additional information to the one held by our musical form which takes part in the generation process.

Another "trivial" transformation would be diatonic transposition. However, it depends on the key of the composition in a very strict manner (in major and minor modes, the degrees might be equal, however, one semitone difference is a dissonant interval). We can only estimate the key [47], so we leave this kind of transformation on the generating model to be possibly learned as a variation.

Another special but not so trivial transformations are those involving an empty fragment (rest-measure), to be labeled with primary "_". These potentially provide a meaningful additional (dis)similarity values for the resulting form or for additional conditioning of the generation models. We chose to omit the potential information gain of additional for the sake of limiting the scope of this work.

Nevertheless, the rest-measures are not to be considered variants of anything and we consider transformations involving them as trivial.

6.5 Transpositions of Nonidentical Fragments

In the straightforward process of determining chromatic transpositions for identical fragments as described in previous section, there lies a hidden problem regarding the coherence of transpositions throughout the composition. The first step, i.e. detection of identity transformations, leaves us with groups of terminals in form, e.g. $(A, 0, 0, ...)$, $(A, 0, 7, ...)$; $(B, 0, 0, ...)$, $(B, 0, -19, ...)$,... For correct utilization of the form, a relation which we call *maximum similarity transposition* is necessary to be determined. Informally, it is intended to be a chromatic interval between nonidentical fragments where the two fragments sound the most resembling. The case of not determining the maximum similarity transpositions between fragments leaves us with a possibility of generating musical pieces with seemingly random long-term and *intertrack* pitch contours. That is because relying on the main three generating submodels 2.3, leaves them with a more complicated task of keeping melodic and harmonic interfragmental pitch relations sensible while having a limited input.

Nevertheless, human composers do not need such information because of their aesthetic sense, their imagination, skills of viewing the composition as a whole. These allow them to compose their works while nonsequentially and repeatedly navigating the harmonic and melodic contours of their composition to be consistent at many different time scales.

6.5.1 Definition of Maximum Similarity Transposition

The maximum similarity transposition between two fragments is the chromatic transposition of the second fragment with the lowest dissimilarity value for all possible chromatic transpositions of the second fragment. It aims to estimate the chromatic transposition (i.e. the previously defined *identity transposition*) between the first fragment and version of the second fragment that has been transformed into the first fragment in transposition invariant fashion.

The transformation of the second fragment can be imagined as it is done in an abstract musical space that has no notion of interfragment transposition. That is by e.g. choosing a representation[5] that keeps the harmonic and melodic specifications and omits encoding of the pitch to make the fragments transposition invariant[6].

However, our pseudometric is working with tonal information which means we get the same dissimilarity results for transpositions modulo octave (it is octave invariant). That is, we can get only results of +0, +1, ..., +11 as transpositions. To detect the correct octave difference to add to our transpositions, we perform a duration-weighted mean of pitches in the two involved fragments, where on the second fragment, we apply the maximum similarity transposition. To find the final "absolute" maximum similarity transposition we find the correct octave by

[5]Such a representation encodes only the intervals between the notes that form both the melody and harmony, much like as specified in authors Bc. thesis [7] p.15 Sec. 3.1.2.

[6]Modern western music tuning systems are roughly transposition invariant[5]

searching the octave space (e.g. -36, -24, -12, 0, +12,...) to be added to the found transposition to minimizes the difference between means. This can result in the found transposition (modulo octave i.e. chromatic interval) to be changed into an inverse chromatic interval. It happens in case it is closer in the mentioned differences in means to subtract an octave e.g. best chromatic interval to add to the second is +11 (i.e. the second fragment is -11 semitones transposed), but the note pitches are "closer" to the first fragment if we add the two octaves below (-24), we get a final absolute transposition of -13 (i.e. the second fragment is 13 semitones transposed).

Nevertheless, for some cases, this method is inconclusive e.g. the simple case of a *dyad* of +0 and +12(+0 in the next octave) chromatic intervals (against the tonic) compared against a note of chromatic tonic interval +6 (against the tonic).

For such cases, we don't know which one of the two notes in the dyad is matched. Arbitrarily, the octave of a lower octave is chosen meaning that the final relative transposition of the second fragment would be -6.

This is, however, a problem of the intention behind maximum similarity transposition, not by it itself, the intention against the strict rules in of pleasant music.

This arbitrary choice of a solution for such ambiguity is the first sight of a developing problem with relative transpositions we describe in the following subsection, which ends up in avoiding these altogether. The problem itself is twofold, while in this case, it is the possibility of extracting a form that guides the fragments (through imposed transpositions) out of the bounds of playable pitches. To our relief, the techniques described in this section will be utilized, just in a different manner.

6.5.2 Relative and Absolute Transpositions

In this section, we first describe problems with relative transpositions of determining the transposition itself, and then the problem with clustering the relatively transposed variants and utilizing them for training of our models.

6.5.2.1 Relative Transpositions

If we look on the transposition as a metric, the transpositions defined between identical fragments (without modulo octave), satisfy all the axioms of a metric. That means we can use them by keeping the relative transposition of each fragment against its first occurrence. However, with variants of a primary, the maximum similarity transposition (i.e. transposition between non-identical fragments) is defined with the use of our pseudometric, and therefore it does not satisfy the triangular inequality, see 5.1.3. Moreover, even if it was not based on the pseudometric, the following case illustrates the problem of triangular inequality in music in general, and this case even illustrate the disadvantage of the *fixed-arbitrary-length* approach of the analysis:

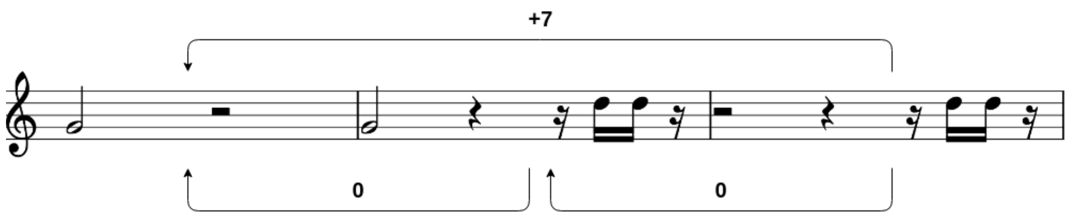


Figure 6.4: Intuitive relative transpositions between three measures, where transpositions cannot be treated like a metric. In terms of maximum similarity transposition, given this transposition, the two fragments sound the most similar. The problem is not only the transposition, but also how to cluster these fragments as the second measure is fairly similar to both of the measures, however the first and the last are not similar. In this case the problem occurs, because we arbitrarily fragment the example by the three measure, however, there are four "meaningful" objects there, where the measure in the middle consists of two objects.

Therefore, determining the correct T part of a terminal in our musical form by relative transpositions is impossible, without changing T into a quadratic array (*fragment a is similar to b but only for transposition x and to c but only for transposition y*).

Furthermore, in the case illustrated in Figure 6.4, the intuitive transposition of +7 would not be found using our similarity pseudometric, because while trying all the possible transpositions for such rhythmically different fragments, all the runs of the similarity matching algorithm would match the rests together and perform *deletion* and *insertion* operations regardless of the pitches of the notes involved, resulting in a randomly chosen transposition value e.g. 0 for our implementation.

6.5.2.2 Relative Transpositions in Form and Models

If we look further, on decisions of how to present pairs of fragments to a variant generating model, there starts to be another problem. Clustering the fragments into variants of each other based on similarity that comes from the best found transposition (maximum similarity transposition), leaves us with large possibility of identifying dissimilar, or for worse part dissonant, pairs under same primary P in the extracted form.

Indeed the abstract boundary of belonging to a variant cluster for primary P might be possibly crossed only by one specific maximum similarity transposition of a fragment against the other. However, this transposition of a given fragment can be different against each other fragments in a cluster of size 3 or more.

This poses a quadratic array of complicated additional information[7], that is not to be ignored both for the extracted form, and the generating models. Utilizing this amount of information is not straightforward[8]. Ignoring this information could leave us with both folds of the problem with relative transpositions, i.e. extracting a form that guides the fragments out out bounds of playable pitch, and generating variants that are possibly not similar at all.

[7]We would need to represent information for each fragment regarding that the fragment is similar enough to be considered a variant if it is transposed by x semitones

[8]E.g. A task for a model to generate a variant on a given fragment, that, if transposed by x semitones is also a variant of a different given fragment, but if transposed by $-x$ is not variant of other given fragment etc.

6.5.2.3 Absolute Transpositions

When we look at finding the absolute transposition of a fragment against some fixed point, we see that this concept is vague at first sight. Therefore, we find the maximum similarity transposition of the analyzed fragment against a **new fragment**.

The new fragment consists of notes that mirror the rhythmic structure of the analyzed fragment and these notes have a an arbitrary fixed pitch. Mirroring the rhythm is done to counter the case illustrated in Figure 6.4 where for the first and the last measure the transposition is "random" due to *insertion* and *deletion* operations for rhythmically distinct fragments, regardless of the pitches of the notes involved.

The pitch of the mirrored fragment to calculate the maximum similarity transposition is fixed across all the new mirror abstract fragments in the composition. We chose the fixed pitch as $C4$ (60) for all the compositions in the dataset. This gives us a clearly defined way of how to obtain unambiguous T for all terminals in the musical form, which we aim to extract. However, these are not the correct non-identical transposition between the fragments, but against the mirrored fragments.

The T values for identical fragments are now also only estimated by maximum similarity transpositions against their mirrors. However, relatively between the identical fragments F_1 and F_2 the difference between the mirror transposition estimations, $(T_1 - T_2)$, is the exact chromatic transposition (defined for identities only).

6.6 Transposition Aware Similarity Analysis

After we have filtered out the uninteresting trivial transformations (the identities and with rest-measure involved), and after we have determined the estimations of absolute maximum similarity transpositions for all fragments, we analyze the fragments that are not trivial transformation 6.4 of at least one other fragment. We perform the similarity analysis in the way described in Section 6.2.

In similarity analysis, we now work with fragments that are all transposed to $T = 0$, to be the most aligned with the $C4$ pitch ($-T$ for all the notes/chords in a fragment). This approach leaves us with slightly skewed pairwise dissimilarities of fragments that are without any additional transposition directly in the analysed composition[9]. That is, we can possibly omit the variants of pairs where, in terms of string matching terminology, their similarity is based on traces between notes less dominant (in terms of duration) than the notes that influenced the absolute maximum similarity transposition against the fixed pitch mirrored fragments.

[9]And leaves us with transpositions of fragments that, in their transposed form, were not even in the analyzed composition

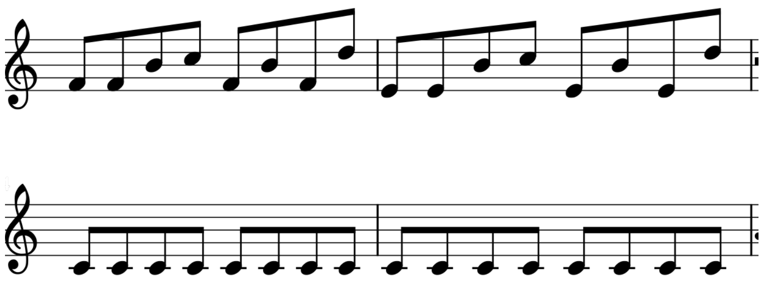


Figure 6.5: Example of a possible omit of two successive variants. The first measure has transposition of +5 against its mirror below it, and the second measure has transposition of +4 against its mirror below it. The two measures have dissonant intervals between themselves, but they are within their key (i.e. using the fairly low weight values from Table 5.1). However, when transposed to be the most aligned with their mirrors and then analyzed together, half of their notes have highly dissonant intervals, Table 5.2, when matched rhythmically.

However, the cost of possibly not finding these variants in the clustering part of the analysis is outweighted by the benefit of avoiding possible out of bounds pitches and dissonant fragments in the generated outputs, or the increased complexity of establishing a more robust system.

6.7 Clustering Fragments to Determine Groups of Variants

The goal of the clustering the is to find groups where each pair of fragments in a group can be considered *similar enough*, within the context of a composition, for the fragments to be considered variants of each other. These groups serve both for determining the musical form and as reservoirs of training pairs for variant and non-variant generating models.

However, even if we have all the pairwise similarity values match perfectly the human perception of similarity, we have no indications of whether the phrases of music occur more than once and what their average intersimilarity is [41].

Clustering the similar measures has to take another criterion into account. That is the similarity is not transitive. If fragment A is similar to fragment B, and B is similar to fragment C, fragments A and C might not be similar because of addition of pitch intervals is not transitive and the similarity metric uses threshold when comparing pitches and durations. The nontransitivity of non-identical transpositions, see 6.5 only worsens the problem.

The boundary of belonging into a same cluster is defined per composition and is influenced by our needs of what we want the extracted form to offer from the information gain point of view i.e. the form might be different from that of human composers, not only by the definition 6.3, but also by purpose.

6.7.1 Choosing an Objective for the Clustering

The direct output of the clustering and the subsequent assignments of P and V for the resulting terminals, however, are not precisely variants of musical material that have musical sense of their own (e.g. *phrases* 1.7.1). The aim of our form is not necessarily to behave like the generally defined form imposed by human composers. We instead cluster the subunits that are forming the meaningful parts of the compositions. As the combined labeling of larger musical units that allows us to distinguish and estimate similarity between them by is carried out by several terminals (labelings of subunits), we aim for such labeling that carries the most information. Therefore, as *phrases* are usually composed of similar members, this aim pushes our requirements to have a relatively high number of clusters.

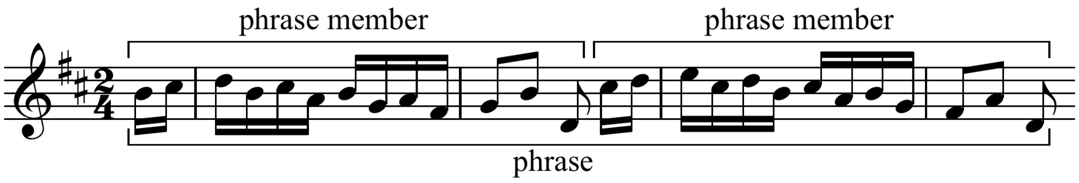


Figure 6.6: A phrase split into two subunits [56]. In this illustraton, the units are logical. However, in our usage, it would be arbitrarily split into 5 subunits according to measures.

Consequentially, high number of clusters means lowering counts of cluster members and increasing the number of outliers for certain clustering algorithms, see [57], [58]. Clusters with low member counts means limited training examples, therefore limited abilities of our variant generating model.

We also aim for clusterings with approximately similar member counts, which inevitably means that intercluster similarities vary a lot for overly monothematic compositions, such as *etudes*. In other words, as opposed to a human analyst, who might conclude that a composition has e.g. one large theme occurring in many variants and a few small dissimilar themes, we aim to identify the building blocks of the themes. As an example, result of an extraction as `A0, A1, A2, A3,...,A42, B0, C0` [10] is unsatisfying.

We chose the clustering $C = C_1, C_2, ..., C_k$ that maximizes the entropy over the counts of members in clusters. For total number of fragments N:

$$H = -\sum_{i=0}^{k} \frac{|C_i|}{N} \log \frac{|C_i|}{N} \tag{6.2}$$

Clustering that maximizes this is, however, trivial, e.g. clustering with n clusters each of 1 member. Such clustering neither helps the form as it is supposed to be a hierarchical overview (e.g. a grammar tree), nor the variant generating model, which will have no training data.

Therefore along with the maximization of entropy, we also require that the

[10] A simplified notation of terminal $\texttt{PV} = (P, V, x, x, x)$

resulting clusters approximately a number of members.

$$H = -\sum_{i=0}^{k} \frac{|C_i|}{N} \log \frac{|C_i|}{N} \quad if \;\; C_i \geq M \tag{6.3}$$

The value of M is chosen to be 5% of the total number of analyzed fragments rounded to the smaller natural number (floored).

6.7.1.1 Meaning of Outliers

As an issue of data mining, outliers are usually considered noise and there is an aim to remove the them as anomalous data [59]. In our application, as the data comes from the human composers considered masters of their art, we consider no fragment of a composition worth discarding however dissimilar it might be from the rest of the data.

The outliers are fragments, that are each in clusters of their own. They are be labeled in the resulting form with $DUMMY$ primary. It is natural to have some of these outliers in the compositions, as they could be e.g. some transitioning measures between two themes or climax measures of the composition. However, we want to limit the number of these outliers. Even though they carry the information to the model for generating dissimilar fragments, however, they limit the scope of data available for the model generating variants of fragments.

Therefore, the set of outliers for a composition is considered a grouping O of its own to have the same informational value as other clusters, as defined in this subsection. Moreover, we add fragments that are in clusters of lower cardinality than M to the count of the grouping O, using the same notation as in previous equation (6.3), we aim to find the clustering C that maximizes H:

$$\begin{aligned} H' &= \sum_{i=0}^{k} \frac{|C_i|}{N} \log \frac{|C_i|}{N} && if \;\; |C_i| \geq M \\ O &= \sum_{i=0}^{k} |C_i| && if \;\; |C_i| < M \\ H &= -H' + \frac{|O|}{N} \log \frac{|O|}{N} \end{aligned} \tag{6.4}$$

The grouping O is used only for clustering information gain evaluation computation in Equation (6.4). That is, the cluster (P) assignments remain as defined by the clustering.

6.7.2 Clustering Algorithm

To cluster the fragments, there are many choices of algorithms that more or less satisfy our aims as defined in previous section. Another complication is that we do not know the number of clusters in advance. Choosing the right algorithm along with its parameters is a complicated task.

Both the more known algorithm K-means [60] and Mixture of Gaussians [60] need to be able to compute means (centroids), and as a consequence cannot operate on a given (dis)similarity matrix of fragments.

One option of how to solve this problem is to use Multidimensional Scaling algorithm [61] which non-linearly approximates a projection of the data into an Euclidean space. Linear options are omitted because the similarity matrix does not satisfy triangular inequality. However, the resulting approximation is neither correct or trivially interpretable and was not used.

Considered clustering algorithm using only the similarity matrix as an input were DBSCAN [57], its hierarchical version HDBSCAN [58] and hierarchical clustering [62].

The first two algorithms also work as outlier detectors [63], possibly enabling us to determine $DUMMY$ primary symbols into the resulting form. However, a disadvantage of HDBSCAN is that it classifies points that are close to the cluster, but not having the right density of other points around them as outliers which is undesirable. Another disadvantage (for our purposes) of both algorithms is that they are based on the *single linkage* [62] principle.

Single linkage performs well on finding cluster that are of non-spherical shapes. However, regardless of the abstract cluster shapes possibly found in our data, it is undesirable do consider two fragments similar only because there is a path of similar transformations in a composition. This property of the two algorithms only multiplies the problem of the pseudometric on which the similarities are based, i.e. not satisfied triangular inequality.

6.7.3 Hierarchical Clustering of the Fragments

We chose to use hierarchical clustering for the task and then use the entropy criterion to choose the number of clusters.

Of the possible linkage options, ward and centroid options are leaved out because of their need to have points in a vector space. Single linkage is left out, because of the reasons specified in the previous section. Average and complete linkages behave sensibly for our purposes.

We use the complete linkage, because it ensures that the (dis)similarity between the most dissimilar two fragments that are considered variants of each other are bounded within the whole composition. Its disadvantage is a possibility of missing out the "true" variants because of their other cluster members that were added in the previous steps of the algorithm are too dissimilar. This offer us the step towards average linkage (a compromise between single and complete linkage).

However, that disadvantage of complete linkage does not limit us from offering enough data for our models, for which we have the certainty that the worst case is not too dissimilar. The use of complete linkage also negates the disadvantage of the pseudometric used to compute the dissimilarity matrix as the "shortcuts" of unsatisfied triangular inequality 5.1.3, are ignored this way.

The concrete parameter selection for the complete linkage hierarchical clustering is done challenging the following aims:

Good inner structure of the clusters, such as meaningful variances of intra-cluster similarities.

Low differences in cluster sizes.

Invariance of cluster's inner density

Proportion of fragments in a cluster against the number of fragments in the composition should approximately equal in the desired output

Number of clusters in total should not be unreasonably large or small

All of these aims are more or less satisfied both by the the choice of the algorithm along with the the following process of determining the algorithm's parameters using the Equation (6.4):

Algorithm 1: Determining the Threshold for the Clustering Method

```
Result: List of labels indicating cluster assignments of the clustering that
        is closest to satisfying the general conditions described in this
        section
search_vals = [0, 0.01, 0.02, ..., maximal dissimilarity value];
best_labeling = [0, 0, ..., 0];
best_entropy = $\infty$;
desired_size_ratio = 0.05;
matrix = Our dissimilarity matrix from similarity analysis;
CompleteLinkageHierarchicalClustering::Returns list of labels indicating
  cluster assignments;
CalculateEntropy::Returns the number specified by Equation (6.4);
LabelOutliers::Returns labels with single-membered clusters labeled as -1;
for i in search_vals do
    l = CompleteLinkageHierarchicalClustering(dissimilarity matrix =
      matrix, desired number of clusters = unspecified number of clusters,
      stop and output the clusters at distance threshold = i);
    H = CalculateEntropy(l = l, M = Floor(|l|· desired_size_ratio));
    if $H >$ best_entropy then
        best_entropy = H;
        best_labeling = l;
    end
end
best_labeling = LabelOutliers(labels = best_labeling);
return best_labeling;
```

Hierarchical clustering has two options on how to get the flat clusters i.e. cut its dendrogram. Both of them require a single parameter to be determined. The first stops the steps of merging (or dividing, depending on implementation) the clusters when the desired amount of clusters was met. The second, which is used in Algorithm 1 stops its steps when certain distance threshold has been met. Algorithm 1 aims to find such threshold to obtain the most desirable clusters.

6.8 Results

In this section we are going to look at the resulting extracted forms and their "correctness". As a dataset for evaluation of musical forms is hard to find and our definition of a musical form slightly differs from the one mainly used by musicians, we are going to analyze the extracted form step by step.

One tool for such analysis is a straightforward addition of a musical form into the compositions in a graphical way. In attachment A.7, there are all the compositions from our dataset provided with “lyrics” for each measure indicating the form terminal in form (P, V, T) inscribed into the *midi* files themselves[11]. An illustration of the attached files can be seen in Figure 6.7. We note that all examples from the dataset in this chapter are transposed to be in C major or C minor key. We also note that all the examples and data have their *legato*’s omitted to simplify the task and keep the interpretation on the player.

[11]Visible in any modern sheet music tool e.g. Musescore [64]

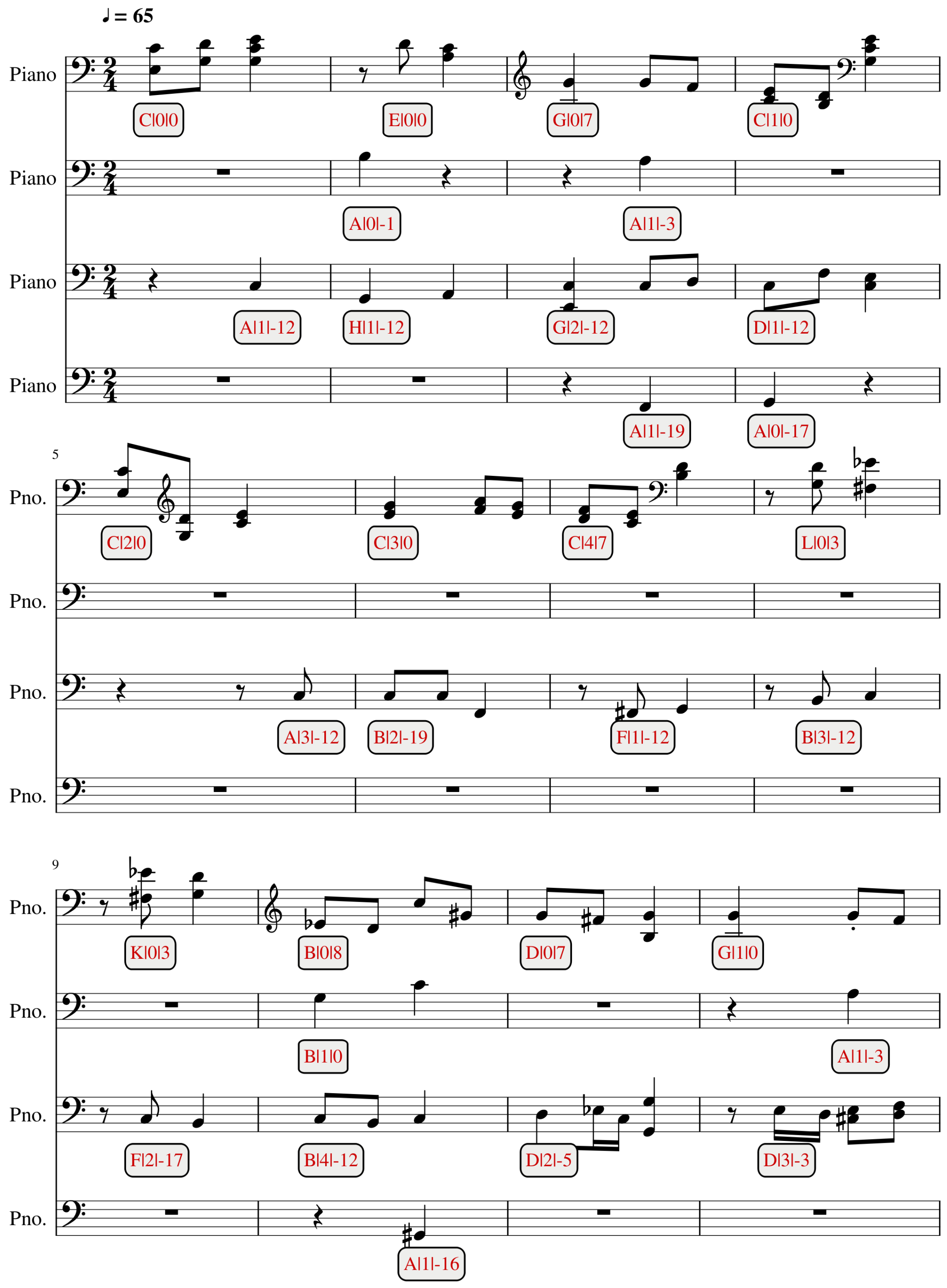


Figure 6.7: The extracted form (P, V, T) inscribed into the analyzed composition (Catches and Glees (12), Hob.XXXIc:16, 2nd mvt. by Franz Joseph Haydn, first 12 measures).

From looking at the inscribed forms, we can check the correctness of the absolute transpositions directly. However, we cannot check the extracted form against the "correct" form the musical form definitions vary and they are vague. The task is mainly about intuitively knowing how the form might look with its vague boundaries for meaningful pieces, and comparing it with the labels found by the algorithm. We can also mainly search for the similar variants and determine if they seem similar by our perception.

Another way how to analyze its results is to look at the details of the clustering and its properties, along with looking at the specific fragments.

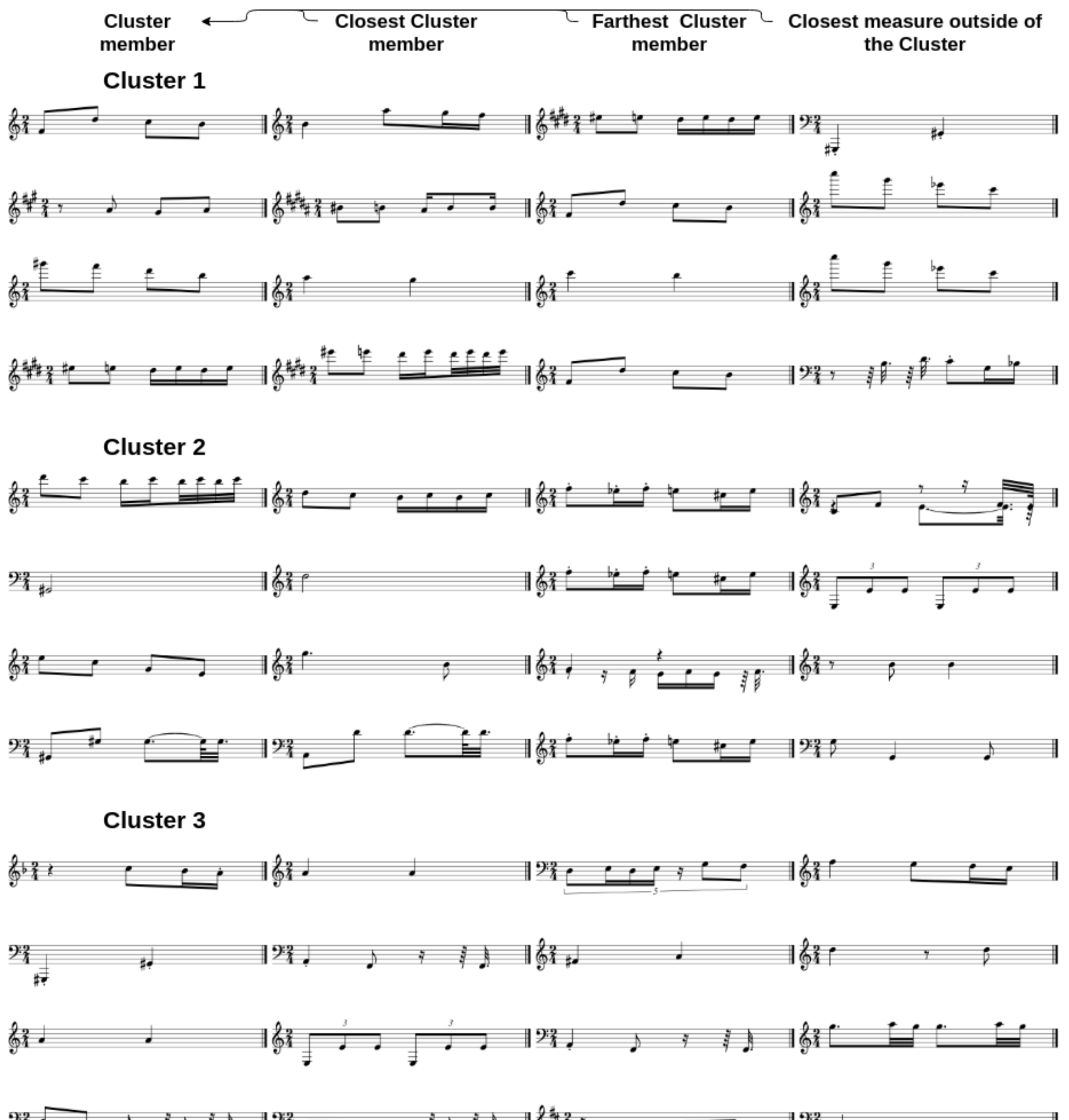


Figure 6.8: Clustering analysis of four samples for each of the three largest clusters in Serenade for Strings in E major, Op. 22, 4th Mvt.

In Figure 6.8, we can see the clustering analysis that shows various cluster members and their closest cluster neighbours, their farthest fellow cluster members and their closest fragment, that is not in the cluster. All the similarity comparisons are made on versions of the measures that are transposed to be the most aligned with their fixed-pitch mirror fragments. That is, the measures in the picture are not necessarily compared as they are, see Section 6.6.

Such analysis is provided for the all the fragments in the three largest clusters for all of the compositions in the dataset, Attachment A.7. It allows us delve deeper into the inner structure of the clusters and to see the advantages and disadvantages of the complete linkage method.

6.8.1 Analysis of the Extracted Forms

From the results, we can see a few main characteristics. When looking at the similarity pseudometric in general, we can see that rhythmically similar fragments are preferred, while still allowing slight variations in the form of fragmentation and small misalignments in rhythm.

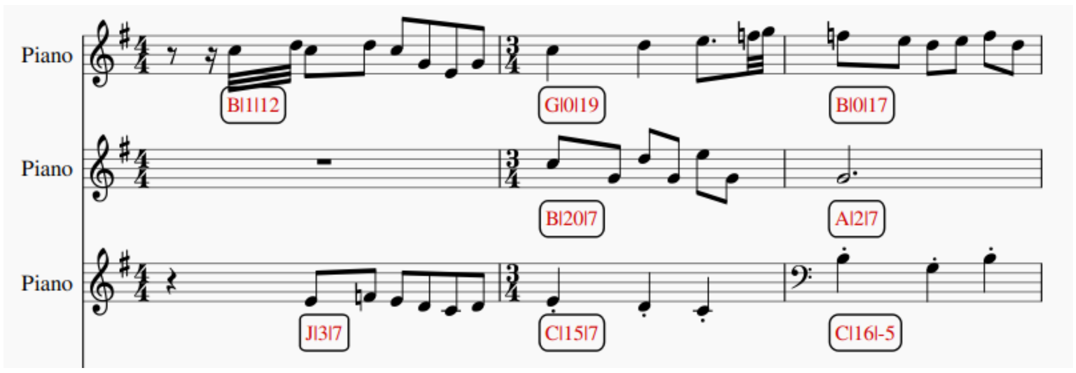


Figure 6.9: Extracted form for the first three measures of Isaac Albéniz - No. 4, Allegro, from Sonata No. 5 Op. 82. Note that probably due to the interaction of pitches and the “careful behavior” of the complete linkage method, the measure on the bottom left and top right did not end up in the same cluster.

In Figure 7.2 *B* variants of 1 and 0 are, in our opinion, correctly determined as variants despite their different total length and slight variation in rhythm. When comparing these two by similarity metric, the 1 variant is transposed down by 5 semitones and all its notes remain their G major mode.

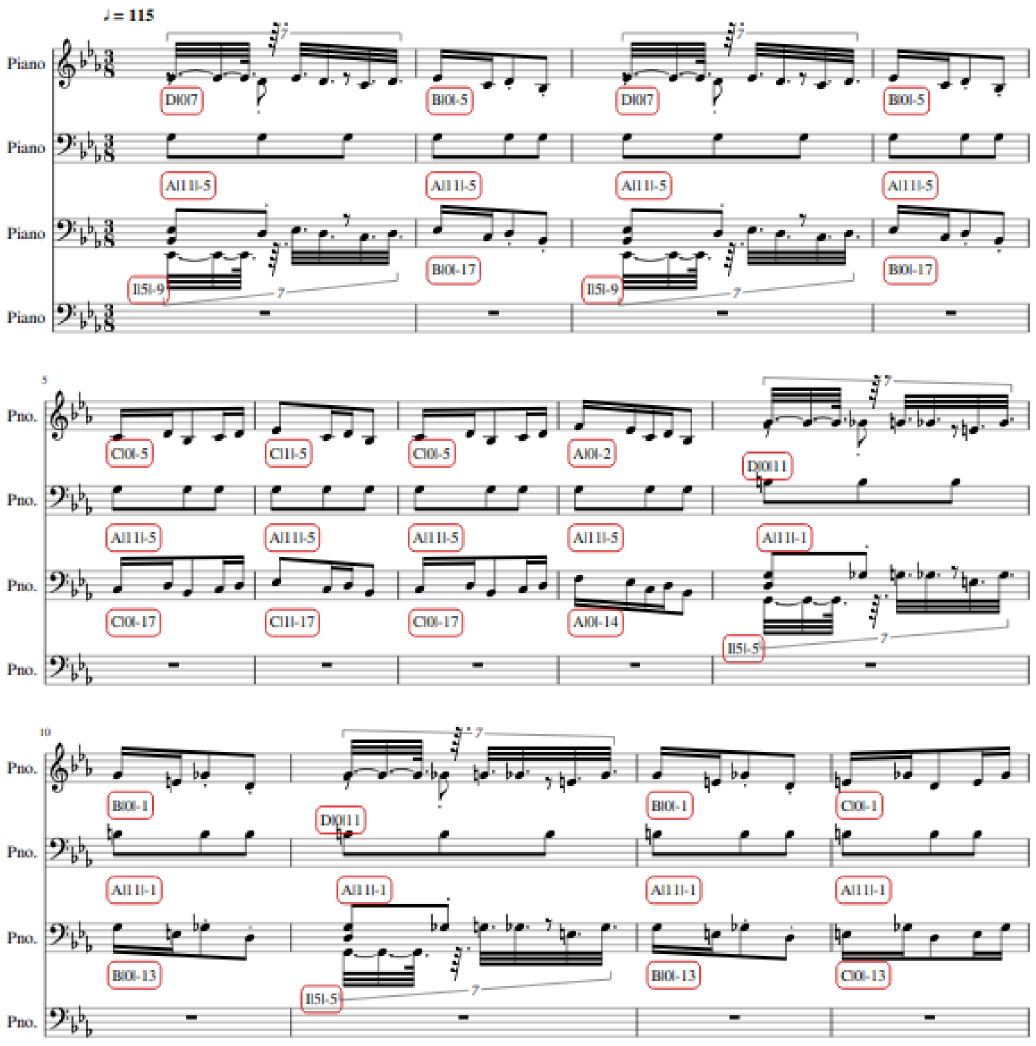


Figure 6.10: Extracted form for measures 1-13 of Isaac Albéniz - No. 6 Op. 47.

In Figure 6.10, we can see that the decision of fragmenting the compositions by measures can reasonably find phrases aligned to measures in higher levels of hierarchical structure, i.e. `D, B, D, B, C` in its first track repeating throughout the composition[12]

[12] The full composition with inscribed form is attached in `cluster_analysis/labeled_midis/misc/minor/albeniz_suite_espanola_47_06_aragon_fantasia_(c)yogore.mid_with_text.mid`

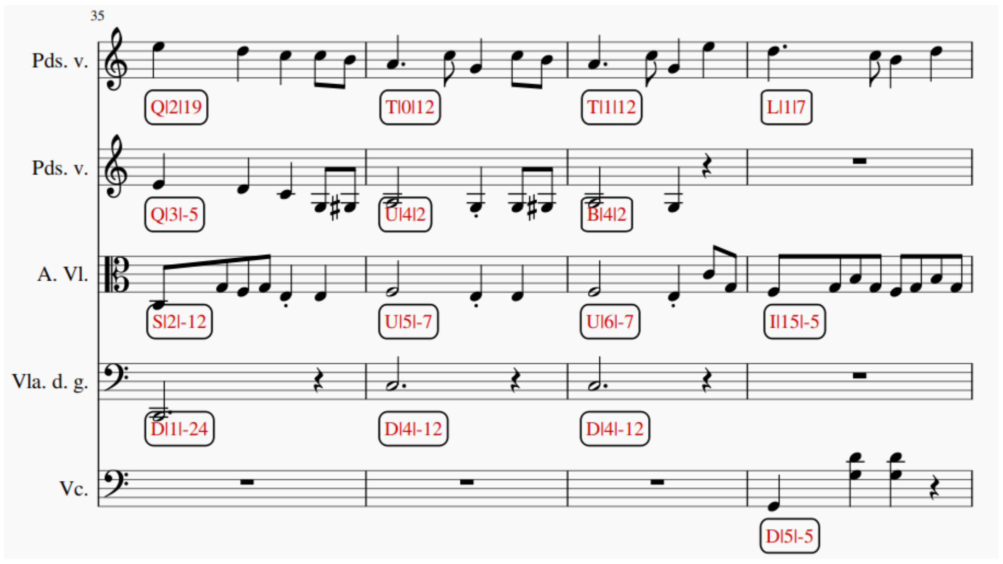


Figure 6.11: Extracted form for measures 35 - 38 three measures of String Quartet No. 10 in E-flat major, D. 87 Op. 125, No. 1

In Figure 6.11, we can see that D variants 1 and 4 should be filtered by the identity filter and not be put into the clustering algorithm together. This is a problem of the dataset, where notes are usually encoded slightly shorter than they are and their duration sometime varies i.e. for both editing software exported midi scores and with recorded performance midi files. We allow slight duration variation in the identity detection preprocess of $0, 1, 2, 3, \ldots$ 32nds for every quarter of the note's duration.

From clustering analysis in Figure 6.8 and the attached clustering analysis A.7, we can see the shortcoming of the *complete linkage* hierarchical clustering method. Where the closest fragment that is not in the cluster often seems more similar than the farthest fragment in the cluster. However, we have the certainty that no too dissimilar pair is clustered. We also see that while *complete linkage* doesn't ensure cluster internal structure other than the worst case, the most similar fragment usually end up in the same cluster.

6.9 Discussion

The resulting form extractions are able to provide the training samples for the models to generate similar and non-similar fragments and to be used as a prescription for the generation process. With having the models trained, even a handcrafted musical form can be used for the generation process.

However the similar measures found may not be conclusive due to the disadvantage of the chosen approach to analyze fragments normalized to be the most aligned with the C4 pitch, which poses inaccurate dissimilarity values. This approach is in the middle of two other approaches. The first is determining the best transposition between all the fragments to not miss out any possible variant which, just as our middle approach, also finds the form based on variants (i.e. transposition) of fragments that are not actually in the data. The second is calculating the similarity between the "unnormalized" fragments and normalizing them only before using them as training for the models. This approach, however, has a similar disadvantage as the chosen middle approach, see Figure 6.5, but surfacing in the generation process (the concatenating and transposing generated fragments as the form prescribes). In the resulting generated composition the non-similar/dissonant fragments are more visible than some omitted pairs during the training and some unmerged, yet similar, primaries in the form.

The clustering part of the analysis could be done differently due to the arbitrarily made decision of the desired cluster size to be around 5 per cent of all the measures in the analysis. However, the entropy goals and the methodology of the clustering seems reasonable. That is, the set goal of having larger number of clusters better finds the implicit musical structure of musical phrases and motives that are larger than a measure e.g. two 4-measure motives, such as, `AABC`, `AABD` are distinguished by it.

A possible future step while having the forms extracted could be using them training a model to be able to produce generated musical forms (in our format), e.g. just like models now generate texts [65].

7. Generation System

This chapter looks further into the models used for generating fragments and into the process of training them and utilizing them in conjunction with the extracted forms from previous chapter.

All the models generate new fragments conditioned on existing fragments. Their purpose is, however, different: one for generating variations, one for generating previously unheard fragments that follow a previous measure, and one for generating previously unheard fragments that should sound at the same time as the conditioning fragment. These different purposes can be served by training the models using different subsets of input/output fragment pairs. While all three models use the same architecture, their different purposes require some differences in parametrization.

This chapter first explains the purposes of each model, then their general architecture, then the general data representation and additional conditioning features and lastly, the model specific architectures along with hyperparameter selection and training specifics.

7.1 Model Purposes

There are 3 models:

- Variant generator
- Accompany generator
- Next-measure generator

Variant generator learns to generate fragments that are similar enough to the input fragment to be considered a variant.

Accompany generator learns to generate fragments that are played at the same time as the input fragment and fit well together with the input fragment.

Next-measure generator learns to generate fragments that are played right after the first of the two input fragments and is not a variant of it. There is also a second input fragment, because of the fact that the first input fragment could be a rest-measure, giving little to no amount of information to the model. Therefore, the second fragment is the first non-rest-measure found in chronological history to the point of interest in the track, if none is found, any non-rest-measure within the track. This model is the main source of "new" musical material.

For two of the three models, variant and next-measure generators, to be trained, we need the preprocessing of raw data into (dis)similar fragments described in Chapters 5 and 6. To clarify the purposes and usage of the models, we show the Figure 7.1 again in this chapter.

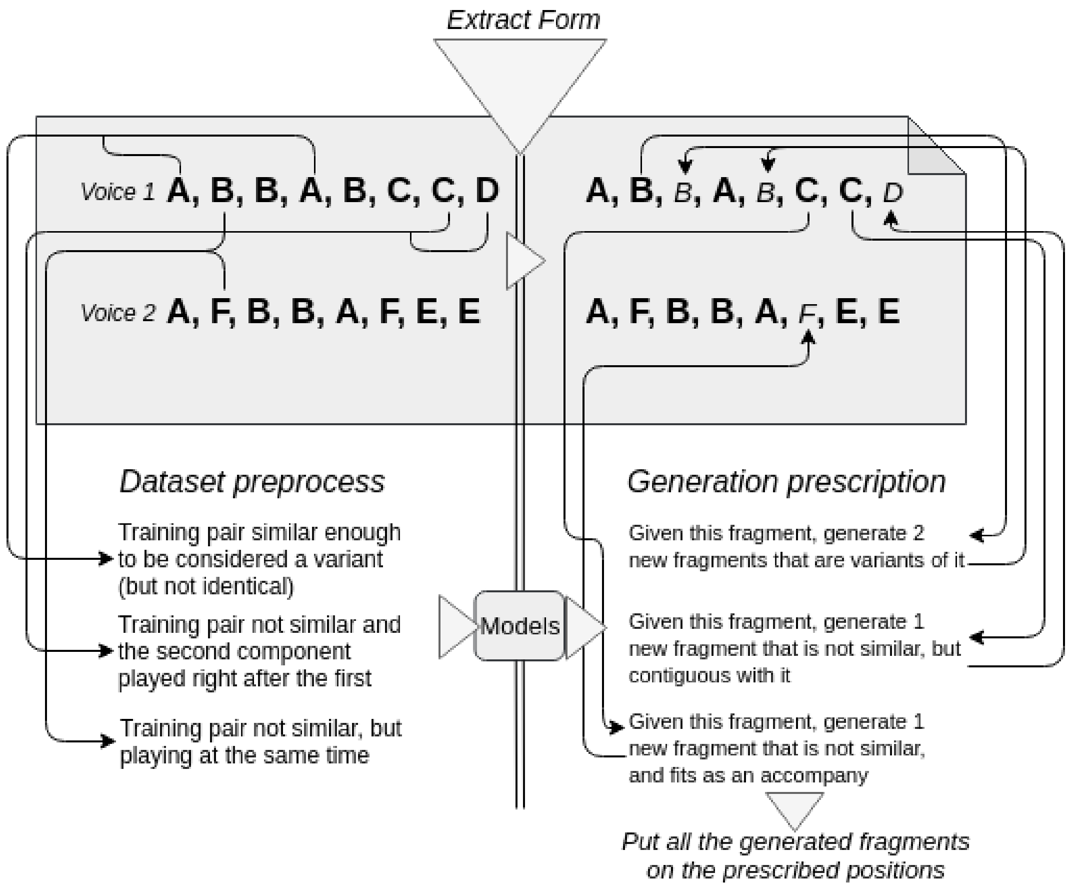

Extract Form
Voice 1 A, B, B, A, B, C, C, D
Voice 2 A, F, B, B, A, F, E, E
A, B, B, A, B, C, C, D
A, F, B, B, A, F, E, E
Dataset preprocess
Training pair similar enough to be considered a variant (but not identical)
Training pair not similar and the second component played right after the first
Training pair not similar, but playing at the same time
Models
Generation prescription
Given this fragment, generate 2 new fragments that are variants of it
Given this fragment, generate 1 new fragment that is not similar, but contiguous with it
Given this fragment, generate 1 new fragment that is not similar, and fits as an accompany
Put all the generated fragments on the prescribed positions

Figure 7.1: Extraction of a musical form provides both the straightforward way to get the meaningful training data and a the way to properly call the models when needed along with prescription for concatenating the generated fragments. In the top-right part of the picture, un-bold letters in italic signify the fragments that have no realization (either seed or generated in previous time step) yet.

7.2 Use of Seq2Seq Models

Seq2Seq models are models that learn the transition function between an input sequence (*source*) and an output sequence (*target*). It consists of two autoregressive generative models, the *encoder* and *decoder*. The encoder encodes the input sequence into a fixed length vector containing the information about the input sequence, and the decoder uses this information to sequentially generate each element of the output sequence alongside with its own vector of information about the “in-process of generating” information about its output sequence.

Most of the current music generation systems use autoregressive models trying to predict the next note(s) using a *state* information about the previous parts of the sequence, see 3.1.

However, our case of generating a fragment given a fragment resembles machine translation tasks, where the seq2seq models are mostly used [66].

7.2.1 Transformer

The transformer introduced in [67] is a Seq2Seq model using *attention* concept that helped improve both performance and training speeds in certain NLP tasks [67], [68], [69] and was previously used for music generation [19] which is the reason for the choice of this model architecture in this work. The attention and self-attention mechanisms allows to model temporal/sequential dependencies without regard to their distance from the currently modeled step in the input or output sequence. Each layer consists of a self-attention layer followed by feedforward layer which is applied to each position in the sequence independently (the parallel speedup [67]).

The attention layer transforms sequence of vectors X of dimension D of the input sequence into sequence Q of queries,K of keys and V of values. Those we chose to be of the same dimensionality as x vectors (they can be smaller or larger) like in [19]. The transformation matrices for the queries, keys and values is split into H attention heads with dimension of the resulting sequence of vectors divided by H and indexed by h. These attention heads allows the model to focus on more parts of the history. Output of the attention layer is a sequence of vectors Z_h for $h \in 0, ..., H$ calculated as:

$$Z_h = Softmax\left(\frac{Q_h K_h^\intercal}{\sqrt{\frac{D}{H}}}\right) V_h \tag{7.1}$$

Then the outputs Z_h are concatenated and linearly transformed to get a sequence of D dimensional vectors Z. For both the encoder we allow the heads to attend to any position (paddings are masked out), whereas in decoder attention layers, there is a mask to ensure that we can attend only to earlier positions during the training because if it learns to attend to the future then during generation the future position are not yet generated and provide no information.

The feedforward layer takes Z and applies two fully connected layers in a time distributed manner (independently). To keep the information about positions of vectors in the sequence to the independent dense layers there is a positional encoding of dimension D added to each of input vectors $x \in X$. This encoding can be some function, usually concatenation of sine and cosine function, or it can be trained alongside with the whole model to learn a positional embedding transformation. In this work, we learned the positional embeddings. Along with learning the positional embeddings, we also learn embbedings of the input sequence units (what is a unit is explained in the next section) to match the hidden dimension size D.

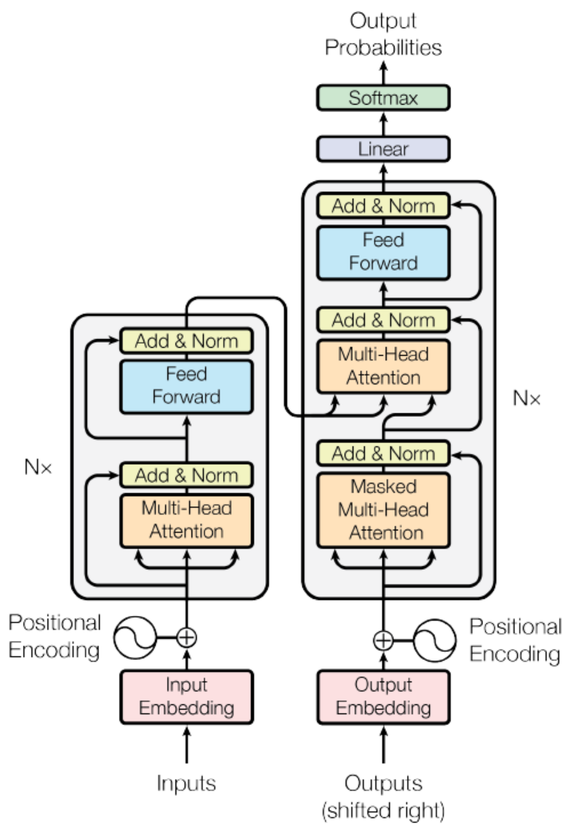


Figure 7.2: Illustration of the transformer from [67].

While the encoder uses only self-attention, the decoder uses self-attention on its input and an encoder-decoder attention (attending on the output sequence of the encoder) in its subsequent layers.

The final sequence of vectors in the last layer of decoder is then stacked into one dimension and there is a linear layer resulting in an output logits vector of the size of a dictionary followed by final softmax layer.

We use the same details regarding residual connections, normalization and activation functions as in [67].

7.3 General Specifications for the Models

In this section we specify the representation of our data and what additional features we use and how we encode them in the input sequence. This representation is shared for all the three versions of the transformer models. We later further define the exact additional features and how they are encoded for each of the models independently.

7.3.1 General Data Representation

We use the representation first presented in the work [18] as a groundwork. We represent a musical fragment as a sequence of decisions: which note to use at a certain point in time, what duration it should have, move to the next point in time etc. We we change the timeshifts from [18] that represent (in centiseconds) a relative time to move the the decisions onward into *durationshifts* that are independent of tempo of the composition. The events used to encode a fragment are:

- 88 note-on events which correspond to ordered midi pitches from 20 to 108. These events start a note.
- 88 note-off events, also corresponding to the midi pitches. These events releases a note.
- 64 durationshift events, each corresponding to a durationshift of 64th notes as one its units. Durationshifts larger than a whole note are split into more durationshifts. These events shift the duration in the fragment. The duration can be directly converted into time if we know the tempo of a composition.
- 16 velocity events. These events represent 16 arbitrarily designed velocity bins. Velocity of these events is applied to all subsequent notes until other velocity event does not change it to other velocity value.

In addition to these events we have three more events. A start-of-sequence (SOS), an end-of-sequence (EOS) and a padding (PAD) event. The first two indicate start and end of a sequence and the padding event works as a standard padding as in e.g. translation tasks [1].

These ranges and the 3 additional events are concatenated resulting in the number of different events of 259, where all the events are represented as one-hot vectors [70]. They are transformed into D dimensional vector by an embedding transformation learned during the training.

Note that all of the fragments are inversely transposed ("normalized") according to the extracted form, see Chapter 6.6. The notes that span across multiple fragments are articulated in each new fragment. Another note is that while the similarity analysis worked with 32th note duration as its smallest unit (smaller notes were omitted), the generation models are working with 64th note duration as its smallest unit.

7.3.2 Additional Conditioning Features

As the design of the models limits them from the dependencies in a larger time frame than the limited size of the input fragments themselves, we introduce conditioning the input and decoder-input fragments with information that is encoded via learnable embeddings.

To add these information into the models, we first make sure the information is categorical and then we add it to the input sequence by concatenation along a new axis. To further pass the additional information to the model, there are several options.

The option of learning an embedding of dimension D and then adding it to the embeddings of the input vector is used in the original model by the positional embeddings. However, when adding more than one information in this manner the model has to distinguish which part of the addition came from which information/feature. This is not necessarily impossible but it is a complicated task even for a human to design unique summation value ranges for float numbers for multiple sources given fixed weights of the models (e.g. design of unix permission numbers for non-binary float values). Therefore, we rather concatenate the information to the input vector while keeping its dimensionality D (the model complexity and memory demands depend of the dimensionality). The keeping of dimensionality consequently means lowering the information space for the main events.

To add the conditioning features to the input (*source*) or decoder-input (*target*) vectors, we lower the dimensionality of the input's main events embedding to a handcrafted ratio between the main events and the embedding of conditioning features. The conditioning features are the same for all the events in the sequence, but must be duplicated because of the independent nature of the transformer's sublayers (i.e. pointwise feedforward sublayers [67]).

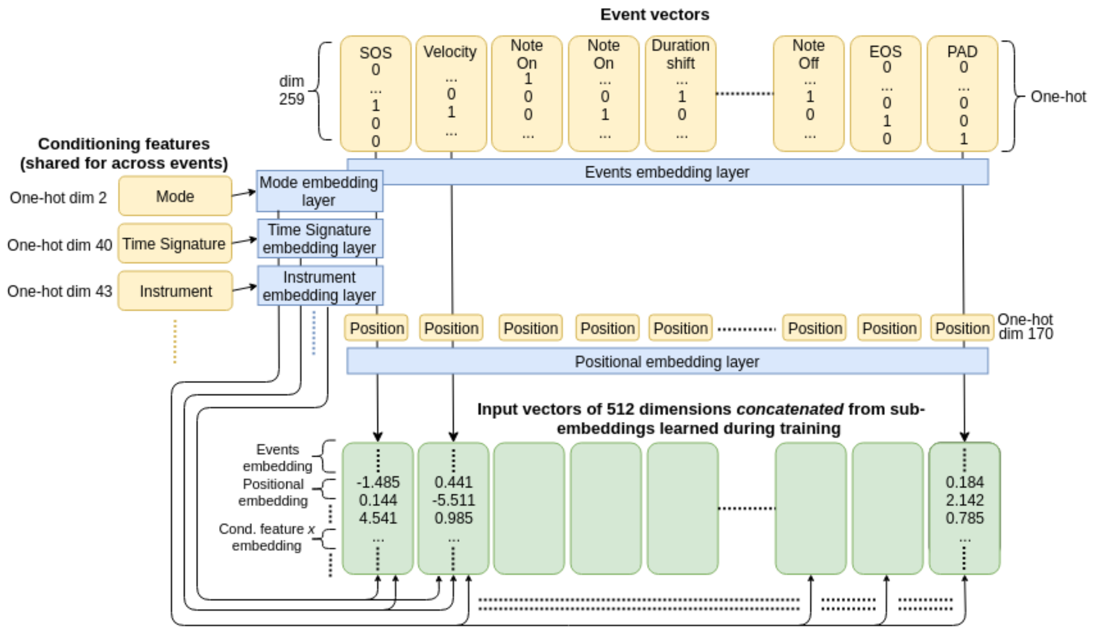


Figure 7.3: Illustration of format of the final input/output (source/target) vectors for the encoder and decoder.

We can also use the extracted form as a storage for the "desired" properties of the "to be generated" fragments. We can add these properties as conditioning features to the models e.g. not only conditioning information, that instrument *a* played the input fragment but also conditioning the model to know that the generated fragment should be played on instrument *b*.

The conditioning features are:

- mode - binary value indicating *major* or *minor* mode to indicate tonality of the desired output
- time signature - categorical value indicating one of the 40 time signatures we work with, see Attachment A.6. The use of this information is for the model to know the beat structure and the total desired duration of the output. This can be encoded to both the input of the encoder and the input of the

decoder (during inference, to the partially completed input/output of the decoder) to indicate that the input fragment is of possibly different time signature than the output fragment.

- instrument - categorical value indicating 43 different instruments or instrument classes, see Attachment A.4. Instrumental information can be also encoded to both the input and “output”[1]. This is useful when input and output fragments are played on different instruments (e.g. accompany model)

- track index - categorical value indicating what index of a voice is the fragment played. We allow maximum 16 voices to be generated and learned upon (it is also a maximum for midi files). In midi files, the ordering of the voices usually holds meaning when the instrument for all or some of the voices is the same. Similarly to the instrument and time signature, we can encode it to both input and “output”.

- motion - categorical value indicating 40 classes of motion. As the fragments have a small scope for analyzing large scale motion, we help this by adding a motion information. The base motion classes are defined on individual fragments. These are “uprising” representing cases when a motion in between every two notes is uprising with maximum of 4 downfalls. Similarly for “downfalling”. If the fragment starts and ends with the same pitch, we encode the motion as a “phrase”. Other motions are encoded as “any”. However, this does not add any new information to the model, so we add the motion of a previous fragment given there is one Enlarging the 4 basic motions into combination of them. However, this does not add the information about the motion over the two fragments i.e. two successive fragments can be uprising, but not continually. Therefore, we add 3 new classes indicating if the difference between means of pitches of the two successive fragments is “U”, lower than -1, “D” higher than 1, or “S” otherwise. As with all the previous features/information, we can add the motion to both input and output to imply the desired motion of the output to be generated.

- number of instruments playing - categorical value of 16 classes indicating how many other voices are currently playing (we know this from the extracted form). While the main value of this information is for knowing if the generated fragment should know if it is meant to be soloing in the composition (solo parts of a composition are usually different in author’s experience), we add the other numbers for voices too, mainly for symphonic compositions where usually when all of instruments play, there is some kind of climax (in author’s view).

- average velocity - categorical value indicating the average velocity of a fragment into 16 arbitrarily binned velocity classes as in the representation of main events 7.3.1. The average velocity of a fragment can not only indicate the desired velocity of an output to not break the velocity coherence of a composition, but also indicate that there should be a climax or a slow down in its rhythm and pitches.

[1]the output meaning described in the previous time signature explanation

- density - categorical value of 170 (max sequence length) values indicating the number of events into sequence. This information is useful mainly for the decoder-input/output in the generation process as it is directly deducible for the encoder.

7.3.3 Hyperparameters

Multiple hyperparameter (parameters not optimized during the learning process) values for the transformer must be set. The hyperparameters not explicitly stated were chosen according to the original [67] paper.

The hidden dimension D is 512. Positionwise feedforward dimension is 2048. Number of attention layers for both encoder and decoder is 6. Number of attention heads is 8. Maximum length of a sequence is 170 due to an arbitrary compromise between the max sized input/output fragments found during the training and memory restrictions on the used (for training) graphics card. Size of the dictionary, alongside with data encoding is specified in the next section. Dropout values are 0.1 as in the original work [67]. For all three versions of the generation models (variants, next-measure, accompany), we used the Adadelta [71] optimizer with default parameters due the unsatisfying results of the optimizer and rate decays from the original work [67] for our purposes [7].

We used gradient norm clipping [72] with a value of 1, to stabilize learning against the vanishing and exploding gradient problem.

Instead of a standard output softmax layer in conjunction with categorical cross-entropy loss, we used stacked sigmoid layers alongside with binary cross-entropy loss. Rather than in order to model each position of the output vector independently, it is due to the shape of the softmax function to which has larger gradients on distributions with large entropy. In our experience, this results in smaller variety of the possible outputs, where we prefer the models to have large variety, e.g. we want to variant generator model to be able to generate many various variants on multiple runs with a same input fragment. To get the final distribution, we normalize the resulting outputs linearly.

As for training parameters like batch size, what constitutes a (custom) epoch in our models, and number of epochs, we didn't find large variety in results for different values. As we only provide the resulting weights and require only the reproducibility of the generated outputs, we worked with batch sizes manually changing throughout the training, due to sharing the computational resources with other people and with a maximum permitted number of 40 batches from a given composition per epoch. We trained the models for largest time possible given the valuable computing resources.

7.4 Model Specifications

This section explains the specifics of the 3 models described in Sec. 7.1 the variants model, the next measure model and the accompany model.

7.4.1 Variant Generator

The variant generator is a model that takes a fragment and generates a fragment similar enough to be considered a variant of the given fragment.

In order to be able to train the model on fragment pairs that are determined to be variants of each other, we therefore need to have the extracted forms available. The model is trained on batches of pairs of various **randomly selected variants of randomly** selected primary symbol (fragment cluster), see sec. 6.3, within one composition. The pairs can be drawn from different voices and/or instruments.

Fragments that are empty measures (the rest is across the whole fragment) are not considered variants. Fragments that have a primary “DUMMY” have no variants and do not participate in the training of this particular model.

Before running the training steps we extract the conditioning features for the fragments. For the *encoder*, we embed the information in the following way:

- We concatenate all the presented embeddings. Dimension of the concatenated embeddings vector is $D = 512$.
- The main events, as described in subsec. 7.3.1, of 259 categories are embedded into 234 dimensions.
- Time signature of the source is embedded into a size of 20.
- Time signature of the target is embedded into a size of 20.
- Instrument of the source is embedded into a size of 40.
- Track number of the source is embedded into a size of 20.
- Mode is embedded into a size of 6.
- Average velocity of the target is embedded into a size of 20.
- Number of instruments playing while the target is played is embedded into a size of 16.
- Density of the target is embedded into a size of 40.
- Motion of the target is embedded into a size of 36.
- Positional encoding (as part of the original transformer [67]) is embedded into a size of 60.

For *decoder*, we use:

- The dimension of all the embeddings concatenated is $D = 512$.
- The main events, see 7.3.1, of 259 categories is embedded into a dimension size of 254.
- Time signature of the target is embedded into a size of 20.
- Instrument of the target is embedded into a size of 40.
- Track number of the target is embedded into a size of 20.
- The rest of the embeddings are identical to the encoder.

7.4.2 Next Measure Generator

The next measure generator also needs the extracted form prior to its training. It takes two input fragments (concatenated), one directly preceding the output fragment, and one additional, which is the first non-rest fragment encountered in the history before the preceding fragment. The additional input fragment is chosen so that when the directly preceding fragment is a rest measure which holds little information about how the following fragment should look, we still have some additional input information. However, we keep the rest measure there as it is an indication for the output fragment that it starts from a long silence.

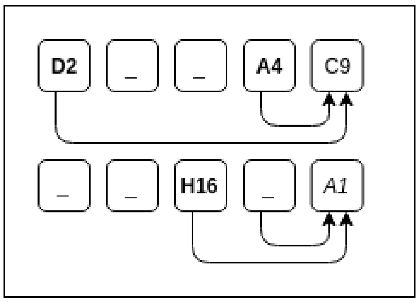


Figure 7.4: The first input fragment is directly preceding and has different primary, the second input is the first non-rest measure found in that precedes the first input fragment

The most important part in training the next measure generator is that the presented successive pairs from the presented training triplets must not be variants of each other. That is, the model learns to generate a fragment which is, to a degree that was chosen by the composer of the training composition, dissimilar to its previous fragment.

Fragments for which the primary is "*DUMMY*" are part of the training data here. Fragment, where the target sequence is a rest measure are omitted from the training.

The conditional features for the encoder and decoder are defined exactly as in the previous section 7.4.1 for the variant generator.

7.4.3 Accompaniment Generator

The accompany generator learns to generate a fragment that is meant to be played *together* with the input fragment. It does not require the extraction of a musical form prior to its training, however, it requires some data structure to hold the conditioning features, for which we used the extracted form.

Conditioning features for the encoder is added in the following way:

- We concatenate all the presented embeddings, dimensionality the concatenated embeddings is $D = 512$.
- The main events, as defined in subsec. 7.3.1, of 259 are embedded into 204 dimensions.
- Time signature of the source/target is embedded into a size of 14.

- Instrument of the source is embedded into a size of 40.
- Track number of the source is embedded into a size of 20.
- Instrument of the target is embedded into a size of 40.
- Track number of the target is embedded into a size of 20.
- Mode is embedded into a size of 6.
- Average velocity of the target is embedded into a size of 20.
- Number of instruments playing while the target is played is embedded into a size of 16.
- Density of the target is embedded into a size of 40.
- Motion of the target is embedded into a size of 36.
- Positional encoding is embedded into a size of 60.

The conditioning features for the decoder is the same as for the encoder.

8. Generation Scheme

Generation scheme is the design of combining combining the calls to the models to generate the final composition. First we generate the first track using variant and next measure models. From a seed measure of a training composition, we generate the first variant. Then we use the next measure model whenever we need a new primary “material”. Every time we get at least one measure with a given primary, every other variant of the primary is generated by the variants model. Next tracks are generated analogously, but we use the accompany model to generate new primary material to ensure the harmonic compatibility with the already generated tracks. If there are rest measures in all of the other measures played at the same time, we use the next measure model to generate the new primary.

Fragments of the composition are not generated sequentially from start to end. They are generated independently of their position within the composition from a starting seed via cooperation between various calls on the models. The starting seed is a fragment from the original composition. However the starting the seed is resampled by the variant model so the generated composition should not have copies of any original fragment.

Finding the proper seed is done by finding the largest cluster (a primary with the most variants) and selecting the most dense fragment in it (dense fragments are usually more informative than e.g. whole notes over the fragment).

It is important to note that while during training, the conditioning features for the source sequence is determined from the original composition, just like the conditioning features for the target sequence. However, during generation, the source conditioning features are determined from the generated realization of a fragment, while the target conditional features are still from the corresponding fragment of the original composition.

The musical form used for generating a piece plays at the same time a role of a useful data structure. The terminals can additionally store whether the fragment that they represent was already generated or not. Once that it was generated, the form stores the generated “realization” of the fragment. As the generated fragments are transposed towards a fixed pitch, the generated fragments are transposed as instructed by the extracted form at the end of the generation process.

Instead of a generated form, one can of course use a handcrafted form. However, to properly handcraft a form that reflects the behavior of our forms as we extracted them from the training data, one would first need to study the form extraction results, so that potential idiosyncrasies of the form extraction process are taken into account. Therefore, for the sake of simplicity, we cancel this effect by directly using as a generation recipe forms that have been extracted from the dataset.

8.1 Design of Model Cooperation

For the overview of the model cooperation during the generation process, we present a simplified view on the whole process in the following figure:

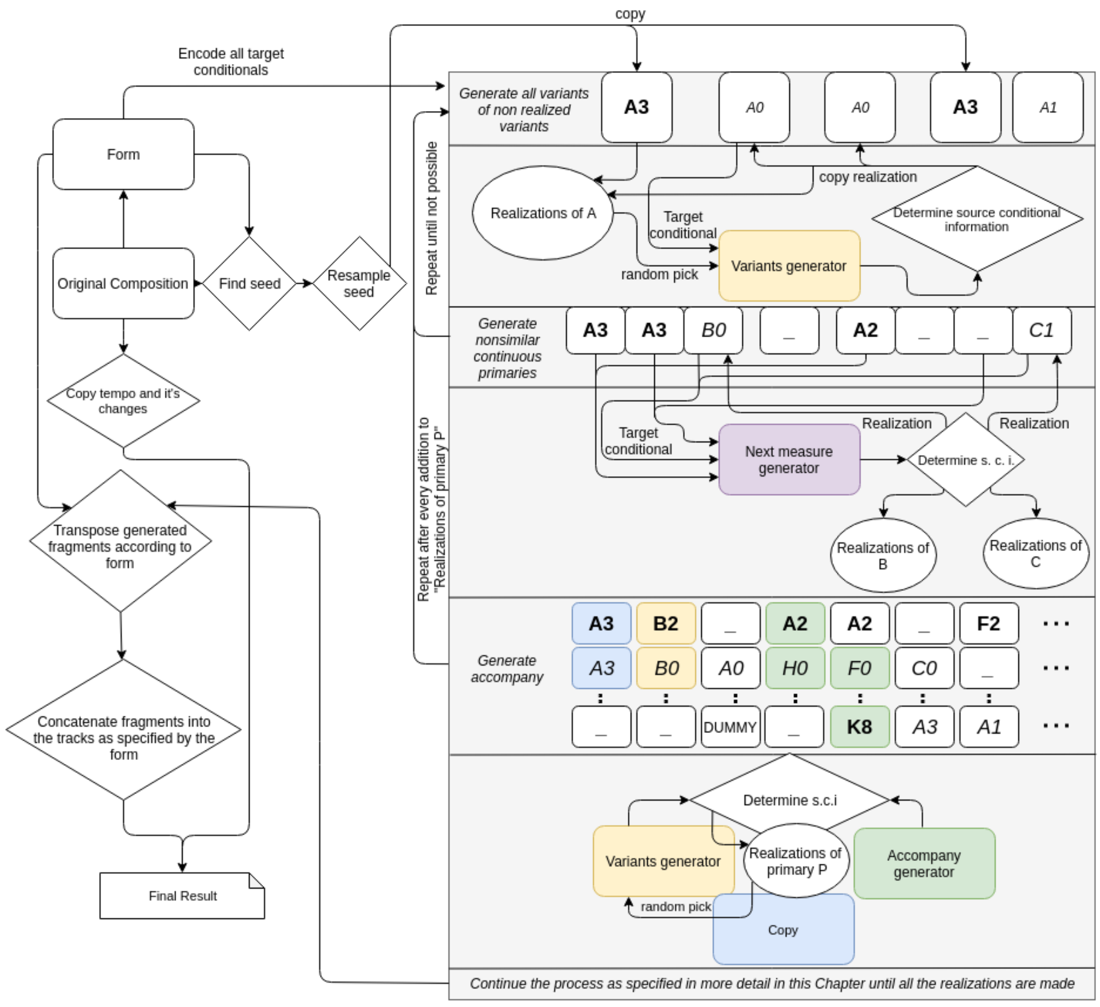

copy
Encode all target conditionals
Generate all variants of non realized variants
A3
A0
A0
A3
A1
Form
Original Composition
Find seed
Resample seed
Repeat until not possible
Realizations of A
copy realization
Target conditional
Determine source conditional information
random pick
Variants generator
Generate nonsimilar continuous primaries
A3
A3
B0
_
A2
_
_
C1
Copy tempo and it's changes
Target conditional
Next measure generator
Realization
Determine s. c. i.
Realization
Repeat after every addition to "Realizations of primary P"
Transpose generated fragments according to form
Realizations of B
Realizations of C
Generate accompany
A3
B2
_
A2
A2
_
F2
A3
B0
A0
H0
F0
C0
_
_
_
DUMMY
_
K8
A3
A1
Concatenate fragments into the tracks as specified by the form
Determine s.c.i
Variants generator
Realizations of primary P
Accompany generator
Final Result
random pick
Copy
Continue the process as specified in more detail in this Chapter until all the realizations are made

Figure 8.1: Illustration of the generation process

The Figure 8.1 can be looked upon from the Original Composition box, from which we extract the form, see 6 and find seed and resample it as explained in section 8.

Afterwards we generate all the variants for the seed according to the form. The source and target sequence conditionals, see 7.3.2 are encoded into the source and target (yet to be generated) sequences stored in the musical form. While the first variant is bound to be generated from the resampled seed, the other variants are generated from a random pick from a list of variant realizations that is created for each primary during the initialization of the generation process. After every generation of a variant, the generated realization is added to the corresponding list of realizations.

When all variants are generated, we look at how to generate new dissimilar primaries in their base variants. We look for a suitable inputs for the run of a next-measure model. After each new generated base primary, We generate all its variants.

When no more base primaries can be generated we use the accompany model to generate accompany fragments regardless of the track of the source or target sequence by the finding the first suitable track in order of the tracks in the musical form for both target and source. After each new base primary generated by the accompany model, we generate all its variants.

When no more accompany fragment can be generated, we use the next measure model in the same fashion as in the previous paragraphs.

There are corner cases which can stop the described procedure from making a realization for every terminal. E.g. a “DUMMY” primary at the start of a track with no accompany played (accompany are full rest measures). These corner cases are generated by the next measure model giving it a full rest measure as first part of its input source sequence and a random pick of any fragment within its track as the second part of its input. If this does not solve the problem, we give the resampled seed measure as the second part of its input.

After we have realizations for every terminal in the extracted form, we transpose these realizations according to the extracted absolute transpositions found in the extracted form. Then we concatenate the fragments into the tracks as specified by the form. The last thing is that we apply the tempo (and its changes) from the original composition and transpose the whole composition randomly by -7, -6, ... or +6 semitones. This final transposition is done to counteract the normalization of the whole dataset into C4 before the training.

8.2 Inference of the Models

Generation process uses the decoder for generating one event at the time. We use a partially greedy decoding with ratio of 0.1 chance for the greedy (argmax) decoding, and 0.9 chance for standard sampling.

For the sampling, we omit the use of softmax function and normalize the resulting logits linearly.

To ensure syntactic consistency of the generated fragments, we check the total duration of the generated fragment after each generated event. If the event is end-of-sequence event and the sum of all the durationshift events in the generated fragment is not enough to fill the required duration given a time signature, we multiply the end-of-sequence probability by 0.95 and resample the current time step, possibly repeating this action. On the other hand, if the event is a durationshift and the sum of all the durationshift events is larger than the required duration given a time signature, we also resample. The second resampling does not alter the probabilities and its drawback is longer resampling.

If we sample a note-off event without a previous note-on event of a same pitch, we also resample the current time step.

9. Generation Results

This chapter shows the results of the generation system itself. For the results of the part for music similarity and extraction of a musical form, we refer the reader to the ending section Results 6.8 of the Chapter 6.

As a whole, evaluation of music generation systems is a problem of its own. Evaluation usually relies on user studies with a "Turing test" question [73], preference tests, [30] or by providing a feedback from professional composers and musicians about some of the example compositions [74]. These often have not been methodologically convincing. The standard evaluation of the more common machine learning tasks via test set accuracy is not applicable because every piece of music is unique and every note/chord in its sequential data has many "correct" decisions on what to play next.

This is a large problem in music generation community (and with other tasks which generate "something") that is slowly getting attention as a robust evaluation can then help design more suitable models.

In this chapter, we first explain the experiments and show the results to show the abilities of the models. Then, we look into the results of an adapted technique from evaluating GANs (Generate Adversial Networks).

To show the abilities of our trained models, we provide several reproducible examples. These are:

- 20 examples of 50 generated variants from one input (source) measure
- 30 composition samples with per measure graphical indications of their intended primary, variant, transposition and an indication of the model from which the measures were generated
- 5 example symphonic compositions to show the generating system's abilities to work with unspecified number of instruments up to 16 simultaneously.
- 5 example extra long (200 measures) compositions to show the generating system's abilities to work with longer sequences[1] and surpass these disadvantages of standard autoregressive models.

9.1 Variants Generator

In this section, we present the resulting outputs from the model that generates a variant measure of a given measure.

[1]Theoretically the length of the compositions can be unlimited, given we have such a musical form (which can be handcrafted).

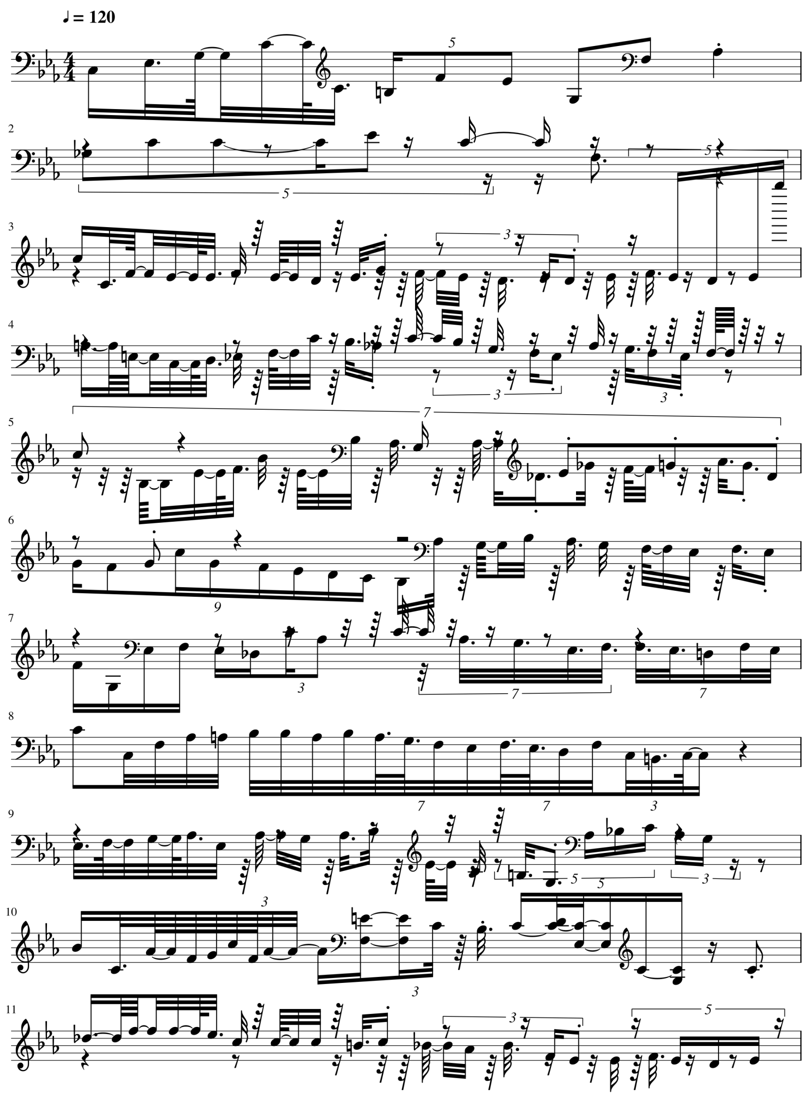


Figure 9.1: First 13 variants of one of the attached 50 variants on the first measure. We use software Musescore [64] for rendering the midifiles into notes. The rendering of notes from plain MIDI is a hard problem and the software sometimes misinterprets the notes creating non-ideal *accidentals*. Another problem with the note rendering is a common inner split of one staff into two (in Musescore up to four) "pseudovoices" which can add unnecessary rests alongside with the played notes as seen in e.g. the second measure in this figure.

In Figure 9.1, we can see example 13 variant measures of the first measure. In Attachment A.8, there are 50 variant measures on the first measure for 20 different first measures i.e. 980 examples of outputs from the variant model.

The resulting measures are more or less similar and from the viewpoint of being pleasant to listen to, they hold their key and are generally rhythmically sound. Some of the outputs from the variant generator can definitely be a nice addition to a human composer's inspiration toolset, while others are less viable.

The main drawbacks of the model is illustrated in the following example:

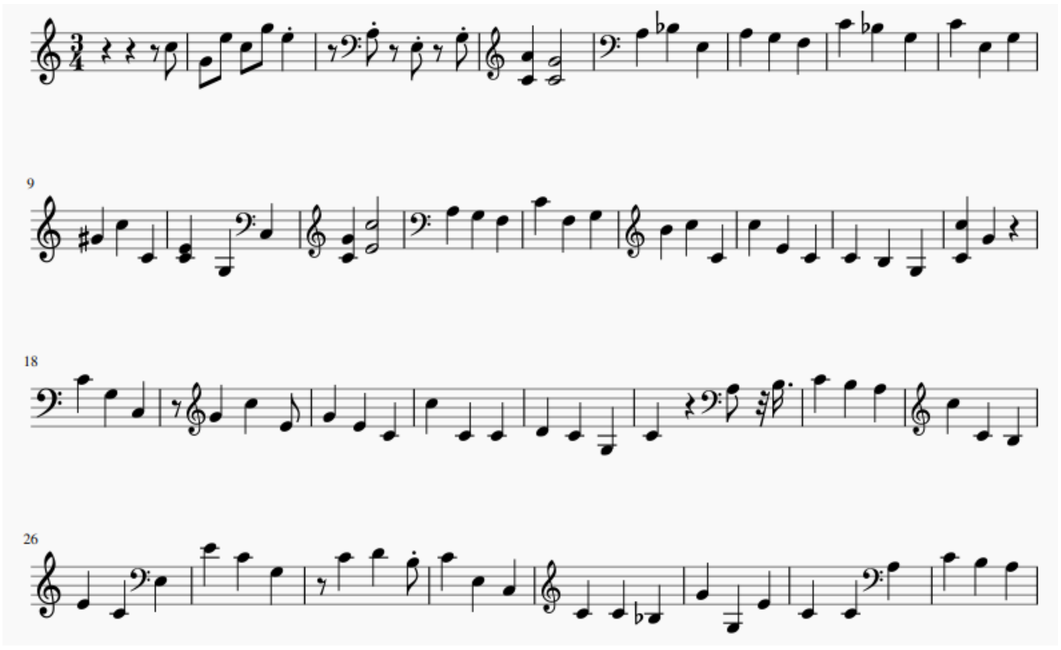


Figure 9.2: Illustration of downsides of the model

In Figure 9.2, while all the generated variants measures are pleasant to ear and similar across the different outputs, their similarity to the first measure, from which were all generated is questionable. From the evaluation of the similarity clustering in 6.8, we know that such clusterings can happen, because rests have relatively low replacement weight. However, one would expect that statistically more measures with similar rhythmical structure to the first measure would be clustered. We hypothesize that this is due to the intra-opus analysis, where measures such as the first measure in the example can be less represented within compositions and forced to be clustered to less similar measures, whereas they could be grouped with similar measures via inter-opus analysis (which is computationally expensive).

From manually analysing the results, the main observation is the high similarity between some of the outputs against lower similarity between the input and the outputs. However, for more dense measures (dense in the number of notes) this problems usually diminishes and the results are more viable.

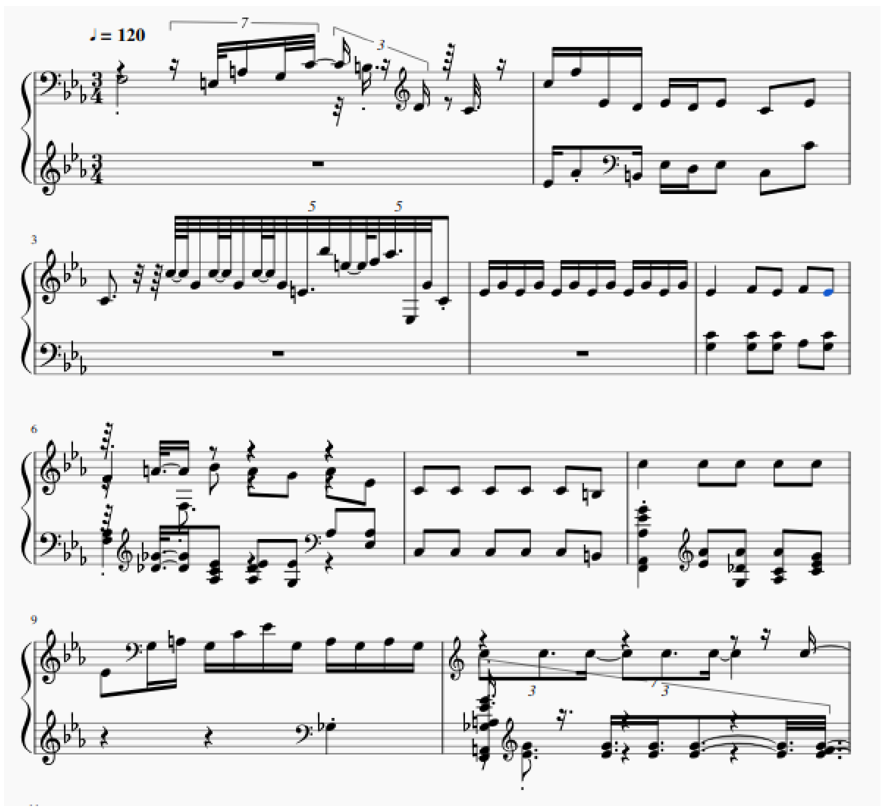


Figure 9.3: Illustration of a possible dissonant output

From the point of view of analysing the quality of the generated measures without regard to their intersimilarity, in Figure 9.3 in measure number 10, we can see a possible downside of the way we decode the raw outputs from the models. While it is common that greedy decoding does not yield the best results as opposed to e.g. beam-search [75], we don't even use the greedy decoding in it full potential. The reason for this is straightforward because we want to keep the variety when it comes to generating variants for a given measure and greedy decoding always gives the same result. Therefore, as described in Section 8.2, at every time step during the sequence generation, we decide with probability 0.1 whether to use the greedy decoding, or 0.9 whether to just sample according to the output probabilities. Increasing this probability to e.g. 0.5 yields more "correct" measures in both their quality and the similarity between the input and outputs of the model analyzed in the previous paragraph. However, it increases the chance to get identical measures. While there is still possibility to enlarge this probability and dump the identical measures, this enlarges the computational cost of generation.

The problem that Figure 9.3 illustrates is that the first chord in the 10th measure is highly dissonant and possibly never seen in training data, which can be caused by simply sampling and not using the greedy decoding, by not having a fully trained model, or most likely by the unusual way that the model was trained i.e. showing multiple desired outputs for one input across the training pairs.

9.2 Generated Compositions

The results of the joint work of the extracted musical form, the variant generating model, the next measure model and the accompany model are presented in this section. Not all the resulting compositions are to be considered good enough. However, or goal was to generate at least some convincing compositions. We also present the less compelling compositions to be able to hypothesise over the reasons for their bad, or sometimes good parts.

In Attachment A.8, we provide:

- 30 samples of length of 70 measures.
- 5 samples of length of 200 measures.
- 5 samples of symphonic type (i.e. up to 16 instruments playing simultaneously) of length of 70 measures.

All these samples are reproducible with the provided seeds. We show some example excerpts from generated compositions. Example of such sample is presented in the following Figure 9.4:

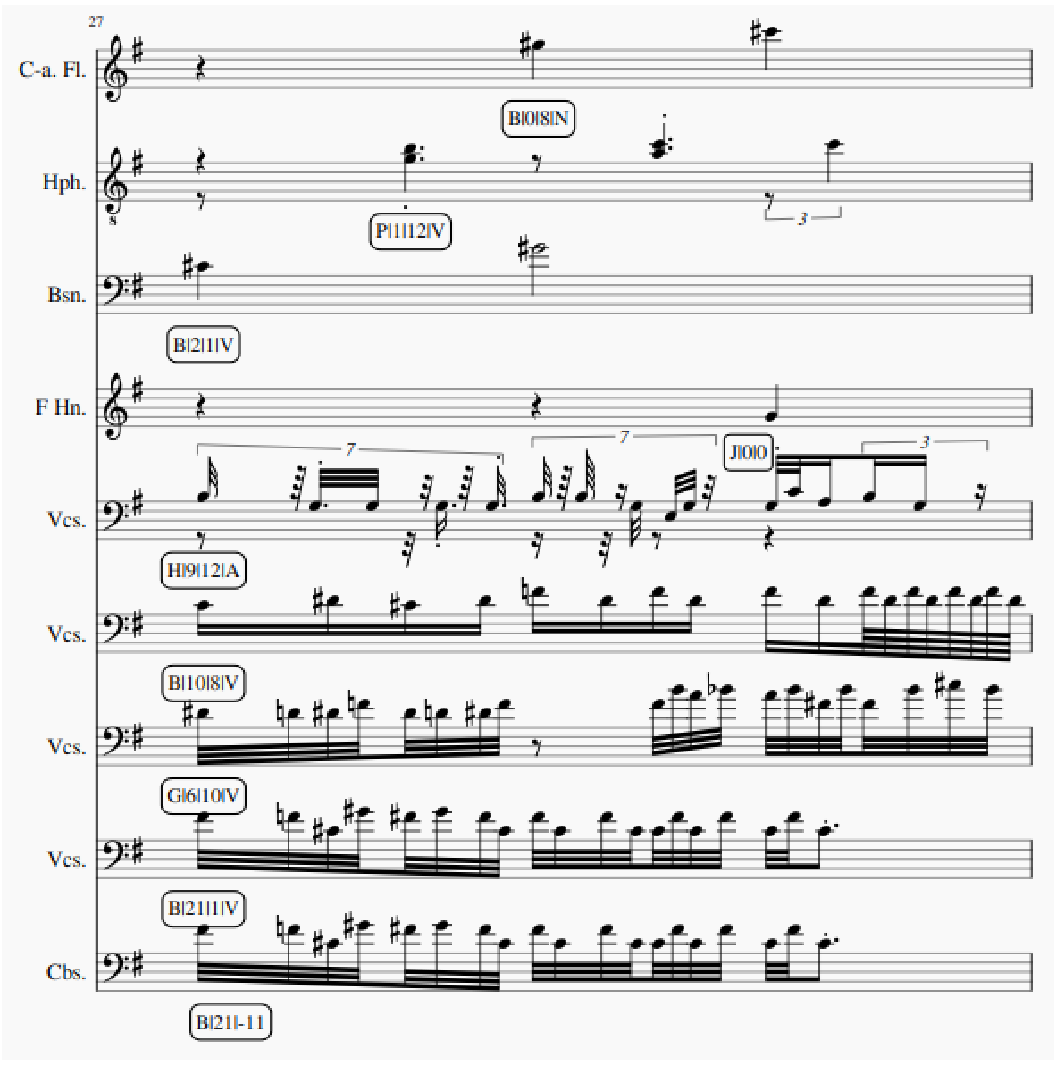


Figure 9.4: Example measure of a symphonic sample.The presented samples have 3-4 information for each measure inscribed in the midifiles. That is the primary, variant, transposition and a possible origin of the measure. The origin is a capital letter, where "V" means the measure was generated by variants model, "N" by next measures model and "A" by accompany model. If there is no origin letter, the measure was copied from another measure having the same primary and variant symbols.

In Figure 9.4, we can see a good example of one of the shortcomings of the system. That is, the harmonic coherence is dependent on more than one fragment, while we only model it with one source. Another problem is the interdependence of the harmony component with time domain, which is also not tackled. Therefore the generated subsequent accompany measures that are placed one after another have no guarantee that they will follow up on continuous dependencies.

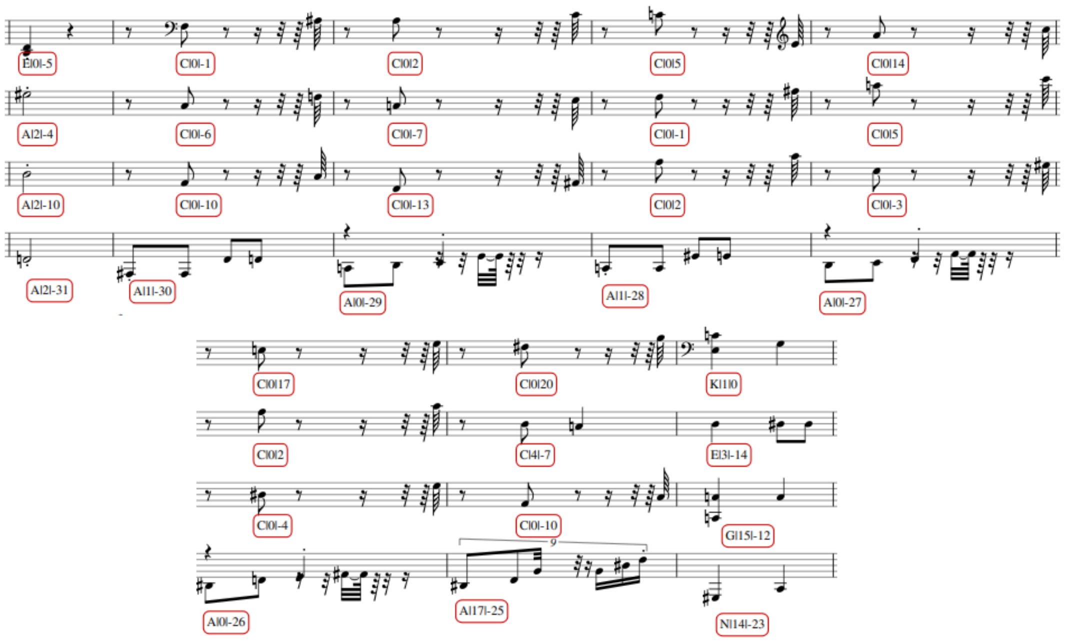


Figure 9.5: Example from sample `10.mid` in `reproducible_samples` attachment, measures 36-43.

In Figure 9.5, we can see how sound the choice to have fragments of measures works to create a gradation of fragments with transpositions of variants `C|0` carried on by the extracted form to the generated composition. This supports the decision of segmenting the compositions into measures.

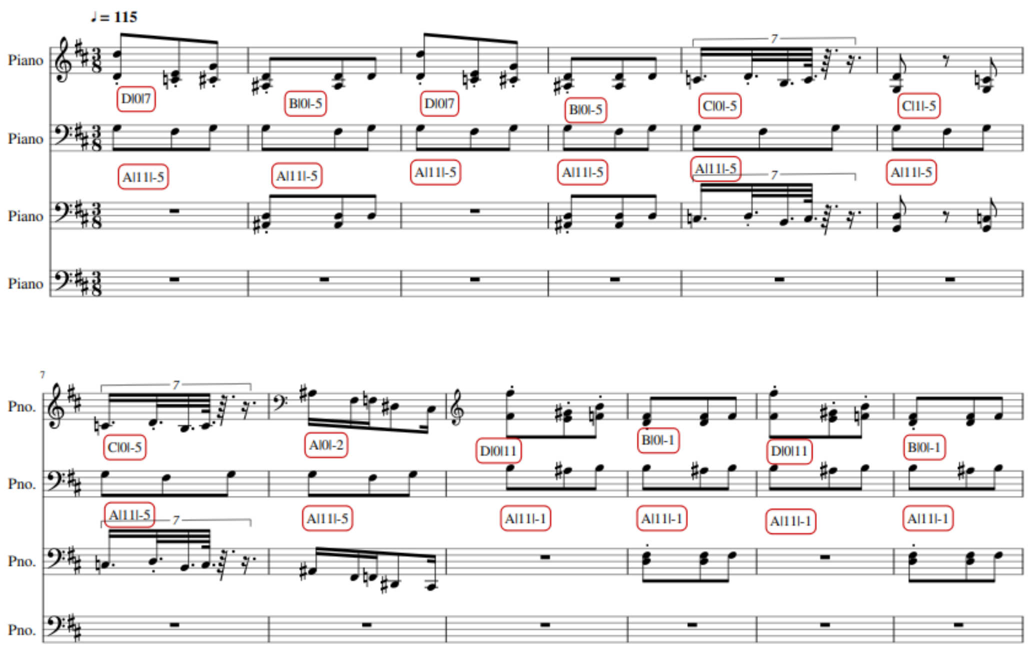


Figure 9.6: Example from sample `38.mid` in `reproducible_samples` attachment, measures 1-12.

Another good example of how the the labeling of measures implicitly captures motives from higher structures of hierarchical levels is the phrase `D0, B0,` `D0, B0, C0, C1, A0` seen in Figure 9.6 repeated 3 times throughout the sample `38.mid` and finishing with another phrase `E0, E1, E0, E1, E2, N1` seen in Figure 9.7. We can also see and compare with Figure 6.10 that is the original composition from which the form was extracted. While using the same form and using a seed measure from 6.10, the generated sample 9.6 is not a copy of its original training composition and it is an interesting composition of its own.

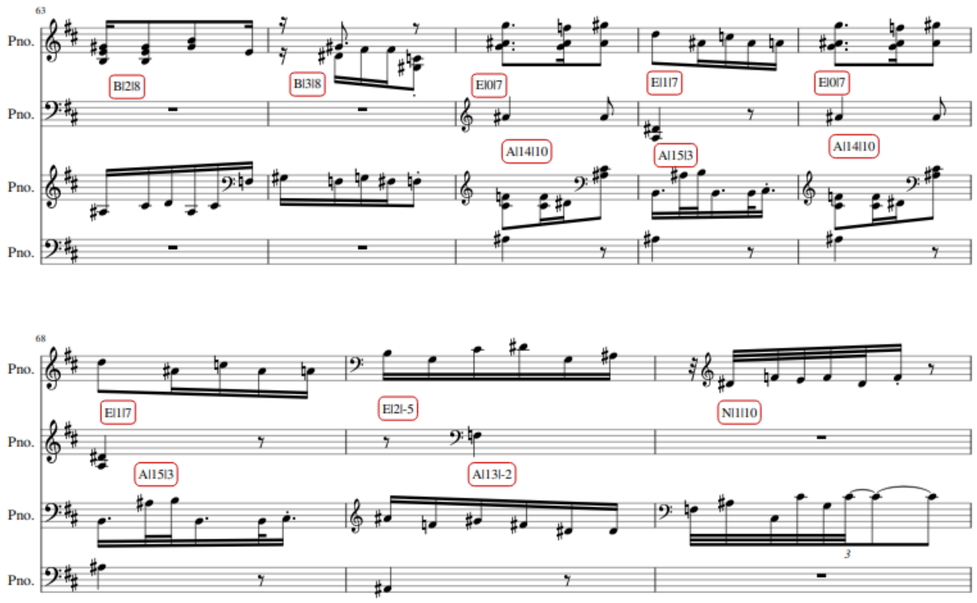


Figure 9.7: Example from sample `38.mid` in `reproducible_samples` attachment, measures 63-70.

The mentioned phrase starting from measure 65 in Figure 9.7 is starting from the first occurrence of primary E, which is generated as a new primary by next-measure model that changes the theme of the composition, but is continuous with the primary B.

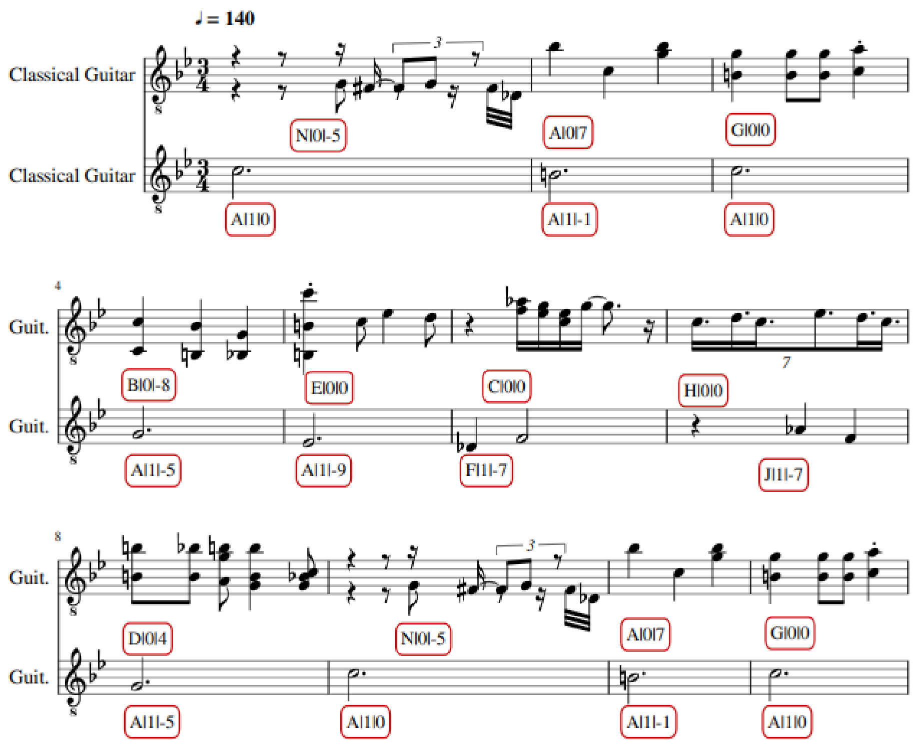


Figure 9.8: Example from sample `36.mid` in `reproducible_samples` attachment, measures 1-10.

In Figure 9.8, we can see the models abilities to learn instruments and track index specifics. As the example well fits the guitar guitar tracks composition.

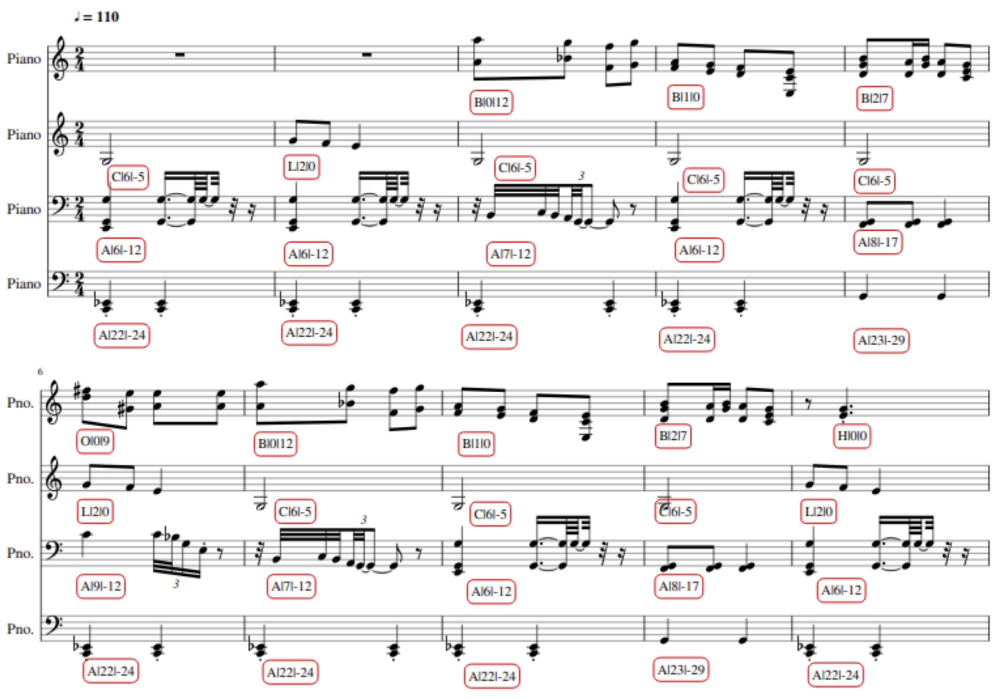


Figure 9.9: Example from sample `13.mid` in `reproducible_samples` attachment, measures 1-10.

In Figure 9.9, we can see how a phrase can be variated by the differing parts of it, see `(B0, B1, B2, O0)` to `(B0, B1, B2, H0)` (later in the same example, measures 29-36 also `(B0, B1, B2, O1)` to `(B0, B1, B2, F1)`) in first track occurring multiple times throughout the sample `13.mid`, or by variating its parts as seen in Figure 9.10 first track i.e. `(B3, B4, B3, N0)`.

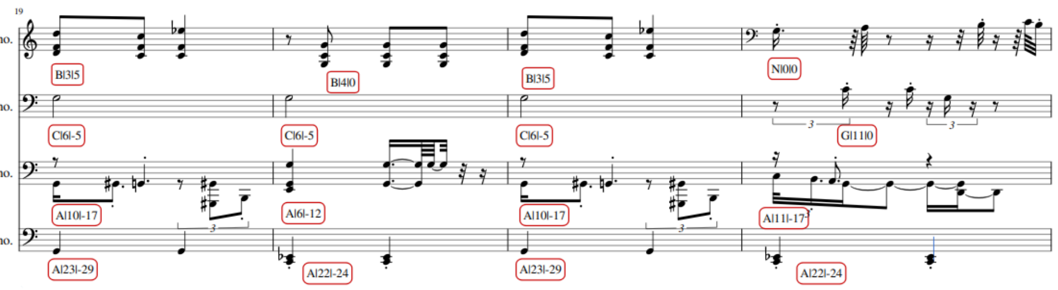


Figure 9.10: Example from sample `13.mid` in `reproducible_samples` attachment, measures 19-22.

9.3 Fréchet Inception Distance

The Fréchet Inception Distance (FID) proposed by Heusel et al. [76] was used to quantify quality of generated samples and compare various GAN types in work by Lucic et al. [4].

It is an attempt to objectively evaluate models that are generating "something". The samples from these models should be from the same subspace as the real data, we aim for a distribution of generated samples that is not distinguishable from the distribution of the real data. However, the problem is that it is hard to compare data points in the space of the original data (e.g. space of images, compositions etc.). Therefore, we need a projection into a space, where the comparison is more viable and the projections maintains a substantial part of expressiveness of the original subspace (or conversely, nonsensical points in the original subspace that does not belong in the manifold of the real data is projected into nonsensical points in the new subspace).

The FID uses Inception Net [77] model model for image classification trained on Image Net to obtain features from one of its layers for the every instance of data. Then, it views these features as samples from multivariate normal distribution and estimates its means and covariance matrix for both the generated data and the real data.

The FID between these estimated Gaussian parameters for the generated g data (μ_g, Σ_g) and the real data r (μ_r, Σ_r) is defined as:

$$FID(r,g) = ||\mu_x - \mu_g||_2^2 + Tr(\Sigma_r + \Sigma_g - 2(\Sigma_r\Sigma_g)^{\frac{1}{2}}) \tag{9.1}$$

The authors show the measure's correlation with visual quality of the examples and with human judgment while outperforming a previous quantitative measure, the Inception Score [78], in robustness against noise.

9.3.1 Adaptation for Music

Inception Net cannot be directly used for music data as it a model for image classification. . Therefore, we experimented with training our own neural network to provide us with meaningful features.

We trained a standard LSTM model to classify the authors of the musical pieces in our dataset. The model is similar in design of the one in author's previous work [7], encodes the current beat given a meter, encodes note pitch plays and their articulations, has dropout of 0.5, but has one hidden layer of size 100.

This model is trained on sections of 100 sixteenth notes with theirs positions within the composition is randomly selected from a composition on each epoch. The tracks of the compositions are merged into one track. The model is trained on the 80% of whole dataset (20% left for its evaluation) and is classifying the subsamples to 42 different authors. The batch size is 4, the number of epochs 3000 with Adam [79] optimizer (default parameters from its paper).

The capacity of the model is much lower than the one used in author's previous thesis ((100) LSTM hidden size against (300, 300, 250) with skip-connections) and it is trained only on a small subportion of the data (randomized sections of 100 continuous notes). This is due to larger dataset and high computational costs. Possibly due to the complexity of the task with 42 authors the limited provided inputs the resulting accuracy is not convincing, at 60% on train and 20% on test sets[2]. However, we mostly care about the internals of network rather than its accuracy. In this experiment, we are interested in whether we can assume that the extracted features capture "something" reasonable to be able to distinguish their authors and styles. Therefore, when assuming that this "learned" subspace is sound, then, when the generated samples are past the real music, it manifests in the "learned" subspace. We select the last state of the LSTM as the embeddings (features) for our generated (and real and random) samples. Then, we compare the estimated distributions via the FID (9.1).

The FID is most commonly used to compare different models as the singular distance between real and generated samples on itself is not indicating anything. We compare the features of the whole real dataset with random subsections of 100 sixteenth notes against the features of random selection subsections of 100 sixteenth notes from 100 reproducible evaluation samples.

To have some kind of a baseline, we look for an upper bound estimate because for the lower bound, we know to expect higher FID than comparison of two subsets of data drawn from the same distribution (FID close to zero). Therefore, we want to know what FID difference is too large. To implement such a baseline, we define a meaningfully randomized generator. The random generator decides which note to play for every sixteenth time frame. It first decides how many notes to play at a given time step with probabilities:

- 0.2 for no note played
- 0.4 for one note played

[2]A similar work [80] is working with different format (non-symbolic data) on a different dataset *artist20* with 20 artists of pop songs, featuring F1 scores of around 0.75 averaged over various timeframes using convolutional recurrent neural network architecture. However, the artists have a strong distinguishing feature here, suitable for non-symbolic (audio) analysis, which is the artists voice, lacking in our symbolic dataset

- 0.2 for dyad played
- 0.1 for chord played
- 0.05 and 0.05 for 4 and 5 notes played simultaneously

The pitch is chosen from random Gaussian distribution with mean being (*upper pitch bound* - *lower pitch bound*) /2 and the standard deviation $\sigma = 20$. The pitch bounds are same for the author classifying model, similarity analysis and the generating models. For notes that are continuing, i.e. longer than sixteenth, it has 1/2 chance to be either articulated or prolonged. The number of these random compositions is 150.

The results of our music adapted version of FID comparison for the generated, real and random samples were following:

The adapted FID	
Generated vs. Real	**5.902**
Generated vs. Random	13.314
Real vs. Random	11.583

Table 9.1: The adapted FID against the real data and the baseline

In Table 9.1, we can see the adapted FID comparison between 100 independent generated samples (attached in `generated_examples/evaluation_generic`) and the training data along with their comparison to the chosen baseline. A lower value of FID means a better result.

In Table 9.2, we show the FID comparisons between various instrumentations of the real and the generated data. This is meant to demonstrate abilities of learned embeddings to distinguish between other target variables than a classification of the author.

The adapted FID for instrumentation of the data	
Piano instrumentation	
Generated Piano vs. Real Piano	**4.650**
Generated Piano vs. Real generic	6.924
Generated Piano vs. Generated generic	2.781
Real Piano vs. Real generic	2.664
Real Piano vs. Random	11.898
Generated Piano vs. Random	14.391
String quartet instrumentation	
Generated String quartet vs. Real String quartet	**3.281**
Generated String quartet vs. Real generic	4.339
Generated String quartet vs. Generated generic	4.448
Real String quartet vs. Real generic	3.320
Real String quartet vs. Random	12.953
Generated String quartet vs. Random	11.184
Symphonic instrumentation	
Generated Symphonic vs. Real Symphonic	**4.805**
Generated Symphonic vs. Real generic	4.798
Generated Symphonic vs. Generated generic	3.024
Real Symphonic vs. Real generic	3.192
Real Symphonic vs. Random	12.997
Generated Symphonic vs. Random	11.186

Table 9.2: The adapted FID for instrumentation of the data

The Table 9.3 shows the FID differences between different instruments for the generated and the real data independently.

The adapted FID for interinstrumental comparisons	
Piano vs. String quartets	
Generated Piano vs. Generated String quartet	1.684
Real Piano vs. Real String quartet	1.372
Piano vs. Symphonic	
Generated Piano vs. Generated Symphonic	4.146
Real Piano vs. Real symphonic	2.419
String quartet vs. Symphonic	
Generated String quartet vs. Generated Symphonic	3.311
Real String quartet vs. Real Symphonic	3.481

Table 9.3: The adapted FID for interinstrumental comparisons. These show the ability of the learned features for author classification to perform reasonably well on distinguishing other target variables (e.g. instrumentation)

We note that while for the Table 9.1, the 100 generated compositions are independent, for Tables 9.2 and 9.3, due to time reasons, the embeddings of the 100 consecutive 16th notes are taken from 30 compositions of 70 measures (approx. 1120 16th notes) for piano and string quartet instrumentation by randomly selecting the notes 5 times per composition. For symphonic compositions, which are the most time demanding for generation, we iterate 20 passthroughs for each of 5 symphonic compositions of 70 measures, meaning the results form symphonic instrumentation are to be believed the least as the samples are not independent.

9.4 Discussion

In this section we discuss the general specifics of the generating system.

As the generated the samples have a specific number of measures, we cannot expect the endings of the compositions to be meaningful. Even if endings are not cut of like that, the models currently have no indication of an ending. The same stands for the beginnings of the compositions, where in our experience it is not that crucial to start a generated composition in a special way.

The fragment generating models are not trained to work as dependent on each other in any order. Therefore, the generated composition can have parts that are not locally coherent. That stands for both continuation and harmony. The harmonic coherence has another drawback, because the accompany model is dependent only on one particular input fragment, where the harmony is dependent on all of the simultaneously playing fragments. This problem is further increased when working with more tracks.

The choice of conditional features 7.3.2 is very hard to properly evaluate as their number is relatively high (8) and they are not independent in between them and in between the 3 trained models. Furthermore, the training of the three main models is time demanding. We only experimented with their choices to create the best sounding results. However, from the resulting examples, we can say that the compositions generally hold their mode, are able to generate by instrumental specifics and generate variants and accompany based on their track order and instrumentation. Conditioning by time signature is also working quite well, because the time signature specifies the total duration of a measure, which is consistent across the generated samples. From examples of variants in 9.1, we can generally see that the density conditioning also has its place in the training process as the variants are usually of similar density. A concept of motion, indication of solo parts or average velocity is hard to even roughly evaluate just from the resulting examples, however, we can say that these are intuitively sensible features to generate fragments with (e.g. a fragment that we expect to be almost silent, downfalling and solo will consequentially have to have its note pitch and note duration choices influenced).

We note that the examples with piano instrumentation are the most consistent and convincing in terms of likeability and hypothesise that this is due to about 50% of our dataset (see 4) is instrumentated like that, while e.g. only 10% are symphonic compositions.

Conclusion

This work has shown a new hybrid approach for music generation that consists of several different subtasks: measuring music similarity, musical form extraction and generation of musical fragments with certain conditions. These three research areas are all rather hard, subjective and focus of ongoing research in the field. Our approach tries to use domain knowledge about music and tries to control the generation process that is usually done by "black box" machine learning models. We show that the approach is able to generate long and coherent multitrack imitations of western classical music compositions.

It is clear we made many more or less arbitrary decisions and the whole system could be made better by further researching each of its main three components individually, e.g. the quality of individual variant fragments generated relies on the quality of the used similarity pseudometric, the quality of the clustering algorithm and the generating model itself. However, the main contribution is that we have shown the approach as a whole is a viable research path and that working with some representation of a musical form, either algorithmically extracted or handcrafted, leads to more long term coherent music.

We used an established music similarity pseudometric based on string comparison and extended it to our dataset with small parameter changes and an extension to cover polyphonic music. This adapted method seems to compute similarity in a way that is good enough for musical form extraction. For extracting the form of musical compositions, we made a decision to arbitrarily segment the compositions by their measures, even though meaningful segments of music are not necessarily aligned to measures, can be shorter or longer than one measure and can overlap. This step simplifies the task and in practice it seems that such choice of segmentation is sound for many compositions, as the labelings of measure-long fragments implicitly determines the inner structure of larger musical phrases and motives. One of the main problems of this approach lies in the inability to find the implicit similarities between motifs of differing overall length in terms of measures. However, the choice of aiming for a larger set of smaller clusters incorporated in the clustering algorithm seems sound for finding the implicit structure of larger motives e.g. `AABC` vs. `AABD` and, consequentially, to keep the training pairs for the variant generator model reasonably similar.

The individual Seq2Seq models we trained (i.e. the variant generator, next measure generator and accompany generator) are producing what we think are satisfactory result fragments when used individually or in conjunction with other models and a musical from to form a whole composition. Even though the choice of conditioning the models to generate fragments of set properties (e.g. mode, instrument, density, solo part etc.) was arbitrary, based on domain knowledge, and not thoroughly individually evaluated, the results did anecdotally show consistency with respect to the conditioned properties. The use of a musical form during generation process reasonably mimics and establishes the desired [12] tension between the surprise and expectation forces in music.

The local coherence between the fragments is attempted by training the accompany and next measure models to create melodically and harmonically complementary fragments. However, the generation of fragments process is sequential (not necessarily in order of the final composition), which can make some fragments incompatible, especially when the local coherence is dependent on more than one fragment (e.g. few fragments into the future, few into the history along with all the tracks playing at the same time), while we only model it independently (on e.g. time domain or harmony domain), and only with one source fragment. The accompany model takes into account only one arbitrary chosen measure, which might not be in harmony with the other measures generated or yet to be generated. This problem enlarges with larger number of tracks so the quality goes down respectively while working with unrestricted polyphony. To tackle the dependence on more fragments, while still (wrongly) assuming independence, we experimented with other models that can be applied after the first “draft” composition is created. This way, the models are able to see both into the future and the history and have all the available tracks already generated. Then, they are used in conjunction and together with variants model along with Gibbs sampling technique [17] [34] to iteratively “repair” the measures to increase their local coherency, while still keeping their long term coherence. However, adding these new models showed less interesting compositions even though the resuls were slightly more coherent.

In future work, we would like to experiment with additional models to be used with musical form that utilize more dependencies between the harmony, continuity and similarity along with Gibbs sampling. Another important research topic is the evaluation of the quality of music generation which is a long standing issue of music generation task.

The resulting system, while of course not producing great quality music on every try, has a good chance to produce good quality, interesting and pleasant compositions. The quality of music also depends on the the chosen instrumentation e.g. piano compositions are roughly 50% of our dataset, while symphonic compositions only about 10%. The FID adapted for music generation shows reasonable numbers for both the evaluation of the resulting compositions themselves and the experiments to show the distinguishing features of the embeddings.

While much can be improved, both in terms of the generation models and the other two major components of the system, the similarity pseudometric and unsupervised form extraction, we believe this work has shown that achieving better control over the long-range structures in generated musical compositions through a learned musical form is a viable path towards engaging musical compositions by a computer.

Bibliography

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[2] Marcel Mongeau and David Sankoff. Comparison of Musical Sequences. *Computers and the Humanities*, 24(3):161–175, Jun 1990.

[3] Julien Allali, Pascal Ferraro, Pierre Hanna, Costas Iliopoulos, and Matthias Robine. Toward a General Framework for Polyphonic Comparison. *Fundamenta Informaticae*, 97, 01 2009.

[4] Mario Lucic, Karol Kurach, Marcin Michalski, Sylvain Gelly, and Olivier Bousquet. Are GANs Created Equal? A Large-Scale Study. *arXiv e-prints*, page arXiv:1711.10337, Nov 2017.

[5] Gareth Loy. *Musimathics: The Mathematical Foundations of Music*. MIT Press, Cambridge, Mass.; London, 2006.

[6] Jan Hajič jr. Handwritten Optical Music Recognition PhD Thesis Proposal. `http://ufal.mff.cuni.cz/~zabokrtsky/pgs/thesis_proposal/jan-hajic-jr-proposal.pdf`, 2017. [Online]; Accessed: 16-06-2017.

[7] Marek Židek. *Generating Polyphonic Music Using Neural Networks*. Bachelor's thesis, Institute of Formal and Applied Linguistics, Faculty of Mathematics and Physics, Charles University, Prague, 2017.

[8] Brian Moseley Kris Shaffer, Bryn Hughes and Robin Wharton. Open Music Theory. `http://openmusictheory.com/`, 2018. Hybrid Pedagogy Publishing, [Online]; Accessed 1-08-2019.

[9] A Complete Guide to Time Signatures in Music. `https://www.musicnotes.com/now/tips/a-complete-guide-to-time-signatures-in-music/`. [Online]; Accessed: 16-12-2019.

[10] Kent D. Cleland and Mary Dobrea-Grindahl. Developing Musicianship Through Aural Skills, unpaginated. Routledge, 2013.

[11] Louis P. DiPalma and David J. Hamilton. Music and Connectionism. *The Journal of the Acoustical Society of America*, 96(2):1218–1219, August 1994.

[12] Michael Cherlin. *Schoenberg's Musical Imagination*. Cambridge University Press, 2007.

[13] Jeff Todd Titon and Timothy J. Cooley. *Worlds of Music: An Introduction to the Music of the World's Peoples*. Schirmer Cengage Learning, Belmont, CA, 2009.

[14] K.H. Eschman. *Changing Forms in Modern Music*. E. C. Schirmer music Company, 1945.

[15] Musical Phrase — Simplifying Theory. Music Theory. `https://www.simplifyingtheory.com/musical-phrase/`. [Online]; Accessed 16-04-2019.

[16] Jean-Jacques Nattiez. *Music and Discourse: Toward a Semiology of Music*. Princeton University Press, 1990.

[17] Gaëtan Hadjeres and François Pachet. DeepBach: a Steerable Model for Bach Chorales Generation. *CoRR*, abs/1612.01010, 2016.

[18] Ian Simon and Sageev Oore. Performance RNN: Generating Music with Expressive Timing and Dynamics. `https://magenta.tensorflow.org/performance-rnn`, 2017.

[19] Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Noam Shazeer, Curtis Hawthorne, Andrew M. Dai, Matthew D. Hoffman, and Douglas Eck. An Improved Relative Self-Attention Mechanism for Transformer with Application to Music Generation. *CoRR*, abs/1809.04281, 2018.

[20] Benoît Meudic. Musical Similarity in Polyphonic Context: A Model Outside Time. In *Colloquium on musical informatics (CIM)*, pages 1–1, Firenze, Italy, May 2003.

[21] D. Rizo. *Symbolic Music Comparison with Tree Data Structures*. PhD thesis, Universidad de Alicante, November 2010.

[22] David A. Stech. A Computer-Assisted Approach to Micro-Analysis of Melodic Lines. *Computers and the Humanities*, 15(4):211–221, 1981.

[23] Zehra Cataltepe, Yusuf Yaslan, and Abdullah Sonmez. Music Genre Classification Using MIDI and Audio Features. *EURASIP Journal on Advances in Signal Processing*, 2007(1):036409, Dec 2007.

[24] Martin Dillon and Michael Hunter. Automated Identification of Melodic Variants in Folk Music. *Computers and the Humanities*, 16(2):107–117, 1982.

[25] Roger B. Dannenberg. A Brief Survey of Music Representation Issues, Techniques, and Systems. `https://www.cs.cmu.edu/~rbd/papers/issues.pdf`. [Online]; Accessed: 25-06-2017.

[26] Sageev Oore, Ian Simon, Sander Dieleman, Douglas Eck, and Karen Simonyan. This Time with Feeling: Learning Expressive Musical Performance. *CoRR*, abs/1808.03715, 2018.

[27] Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W. Senior, and Koray Kavukcuoglu. WaveNet: A Generative Model for Raw Audio. *CoRR*, abs/1609.03499, 2016.

[28] Y. Bengio N Boulanger-Lewandowski and P. Vincent. Modeling Temporal Dependencies in High-Dimensional Sequences: Application to Polyphonic Music Generation and Transcription. In *Proceedings of the 29th International Conference on Machine Learning (ICML)*, Edinburgh, Scotland, UK, 2012.

[29] Jurgen Schmidhuber Douglas Eck. A First Look at Music Composition using LSTM Recurrent Neural Networks. Technical report. [Online]; Accessed: 29-04-2017.

[30] Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang. MuseGAN: Multi-track Sequential Generative Adversarial Networks for Symbolic Music Generation and Accompaniment. *CoRR*, abs/1709.06298, 2017.

[31] Adam Roberts, Jesse Engel, Colin Raffel, Curtis Hawthorne, and Douglas Eck. A Hierarchical Latent Vector Model for Learning Long-Term Structure in Music. *CoRR*, abs/1803.05428, 2018.

[32] Daniel D. Johnson. Composing Music With Recurrent Neural Networks. `http://www.hexahedria.com/2015/08/03/composing-music-with-recurrent-neural-networks/`, 2015. [Online]; Accessed: 29-04-2017.

[33] Daniel D. Johnson. Generating Polyphonic Music Using Tied Parallel Networks. In *Computational Intelligence in Music, Sound, Art and Design*, pages 128–143. Springer International Publishing, 2017.

[34] Cheng-Zhi Anna Huang, Tim Cooijmans, Adam Roberts, Aaron C. Courville, and Douglas Eck. Counterpoint by Convolution. *CoRR*, abs/1903.07227, 2019.

[35] Sander Dieleman, Aäron van den Oord, and Karen Simonyan. The Challenge of Realistic Music Generation: Modelling Raw Audio at Scale. *CoRR*, abs/1806.10474, 2018.

[36] Tomáš Pavlín. *Music Composition Based on a Programming Language*. Bachelor's thesis, Department of Software Engineering, Faculty of Mathematics and Physics, Charles University, Prague, 2018.

[37] Robert A. Wagner and Michael J. Fischer. The String-to-String Correction Problem. *J. ACM*, 21(1):168–173, January 1974.

[38] Walter B. Hewlett and Eleanor Selfridge-Field. Melodic Similarity: Concepts, Procedures, and Applications Computing in Musicology; 11. `https://epdf.tips/melodic-similarity-concepts-procedures-and-applications.html`, 1998. [Online]; Accessed 16-04-2019.

[39] Alexandra Uitdenbogerd and Justin Zobel. Melodic Matching Techniques for Large Music Databases. In *Proceedings of the Seventh ACM International Conference on Multimedia (Part 1)*, MULTIMEDIA '99, pages 57–66, New York, NY, USA, 1999. ACM.

[40] Namunu Maddage. Automatic Structure Detection for Popular Music. *IEEE MultiMedia*, 13:65–77, 01 2006.

[41] Roger B. Dannenberg and Ning Hu. Discovering Musical Structure in Audio Recordings. In *Music and Artificial Intelligence*, pages 43–57. Springer Berlin Heidelberg, 2002.

[42] Jonathan Foote, Matthew Cooper, and Andreas Girgensohn. Creating Music Videos Using Automatic Media Analysis. In *Proceedings of the Tenth ACM International Conference on Multimedia*, MULTIMEDIA '02, pages 553–560, New York, NY, USA, 2002. ACM.

[43] Mark A. Bartsch and Gregory H. Wakefield. Audio Thumbnailing of Popular Music Using Chroma-based Representations. *IEEE Transactions on Multimedia*, 7:96–104, 2005.

[44] Oriol Nieto. Unsupervised Music Motifs Extraction. `http://urinieto.com/NYU/ML/OriolNieto-FinalProject.pdf`. [Online]; Accessed 19-12-2019.

[45] Ron J. Weiss and Juan Pablo Bello. Identifying Repeated Patterns in Music Using Sparse Convolutive Non-negative Matrix Factorization. In *Proceedings of the 11th International Society for Music Information Retrieval Conference, ISMIR*, pages 123–128, Utrecht, Netherlands, August 9-13, 2010.

[46] MIDI Manufacturers Association (MMA). `https://www.midi.org/specifications`. [Online]; Accessed 1-08-2019.

[47] David Temperley. What's Key for Key? The Krumhansl-Schmuckler Key-Finding Algorithm Reconsidered. volume 17, pages 65–100. University of California Press, 1999.

[48] Michael Scott Cuthbert and Christopher Ariza. music21: A Toolkit for Computer-Aided Musicology and Symbolic Music Data. In *Proceedings of the 11th International Society for Music Information Retrieval Conference, ISMIR 2010*, pages 637–642, Utrecht, Netherlands, August 9-13, 2010.

[49] Frederic P. Miller, Agnes F. Vandome, and John McBrewster. *Levenshtein Distance: Information Theory, Computer Science, String (Computer Science), String Metric, Damerau-Levenshtein Distance, Spell Checker, Hamming Distance.* Alpha Press, 2009.

[50] Amos Tversky. Features of Similarity. *Psychological Review*, 84(4):327–352, 1977.

[51] David Meredith. Computer-aided Comparison of Syntax Systems in Three Piano Pieces by Debussy. *Contemporary Music Review*, 9(1-2):285–304, 1993. Paper id:: http://dx.doi.org/10.1080/07494469300640511.

[52] Rainer Typke. *Music Retrieval based on Melodic Similarity.* PhD thesis, Utrecht University, Department of Computer Science, 2007.

[53] Lyle Ramshaw and Robert E. Tarjan. On Minimum-Cost Assignments in Unbalanced Bipartite Graphs. 2012.

[54] R. Jonker and A. Volgenant. A Shortest Augmenting Path Algorithm for Dense and Sparse Linear Assignment Problems. *Computing*, 38(4):325–340, Dec 1987.

[55] Craig Wright. *Listening to Music.* Cengage Learning, jan 2013.

[56] Benward & Saker. Music: In Theory and Practice, Vol. I, p.113. Seventh Edition. ISBN 978-0-07-294262-0, 2003.

[57] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A Density-based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, pages 226–231. AAAI Press, 1996.

[58] Leland McInnes, John Healy, and Steve Astels. hdbscan: Hierarchical density based clustering. *The Journal of Open Source Software*, 2(11), mar 2017.

[59] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer New York, 2009.

[60] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning: With Applications in R*. Springer Publishing Company, Incorporated, 2014.

[61] Trevor F. Cox and M.A.A. Cox. *Multidimensional Scaling, Second Edition*. Chapman and Hall/CRC, 2nd edition, 2000.

[62] Stephen C. Johnson. Hierarchical Clustering Schemes. *Psychometrika*, 32(3):241–254, 1967.

[63] Arthur Zimek and Erich Schubert. *Outlier Detection*, pages 1–5. Springer New York, New York, NY, 2017.

[64] Maxwell Shinn. *Instant MuseScore*. Packt Publishing, 2013.

[65] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 2019.

[66] Denny Britz, Anna Goldie, Minh-Thang Luong, and Quoc V. Le. Massive Exploration of Neural Machine Translation Architectures. *CoRR*, abs/1703.03906, 2017.

[67] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. *CoRR*, abs/1706.03762, 2017.

[68] Thi-Vinh Ngo, Thanh-Le Ha, Phuong-Thai Nguyen, and Le-Minh Nguyen. How Transformer Revitalizes Character-based Neural Machine Translation: An Investigation on Japanese-Vietnamese Translation Systems. *CoRR*, abs/1910.02238, 2019.

[69] Qiang Wang, Bei Li, Tong Xiao, Jingbo Zhu, Changliang Li, Derek F. Wong, and Lidia S. Chao. Learning Deep Transformer Models for Machine Translation. *CoRR*, abs/1906.01787, 2019.

[70] Francois Chollet. *Deep Learning with Python*. Manning Publications Co., Greenwich, CT, USA, 1st edition, 2017.

[71] Matthew D. Zeiler. ADADELTA: An Adaptive Learning Rate Method. *CoRR*, abs/1212.5701, 2012.

[72] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. Understanding the Exploding Gradient Problem. *CoRR*, abs/1211.5063, 2012.

[73] Chris Donahue, Huanru Henry Mao, Yiting Ethan Li, Garrison W. Cottrell, and Julian McAuley. LakhNES: Improving Multi-instrumental Music Generation with Cross-domain Pre-training. *CoRR*, abs/1907.04868, 2019.

[74] Sageev Oore, Ian Simon, Sander Dieleman, Douglas Eck, and Karen Simonyan. This Time with Feeling: Learning Expressive Musical Performance. *Neural Computing and Applications*, November 2018.

[75] Markus Freitag and Yaser Al-Onaizan. Beam Search Strategies for Neural Machine Translation. *CoRR*, abs/1702.01806, 2017.

[76] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, Günter Klambauer, and Sepp Hochreiter. GANs Trained by a Two Time-Scale Update Rule Converge to a Nash Equilibrium. *CoRR*, abs/1706.08500, 2017.

[77] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the Inception Architecture for Computer Vision. *CoRR*, abs/1512.00567, 2015.

[78] Tim Salimans, Ian J. Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved Techniques for Training GANs. *CoRR*, abs/1606.03498, 2016.

[79] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *CoRR*, abs/1412.6980, 2014.

[80] Zain Nasrullah and Yue Zhao. Music Artist Classification with Convolutional Recurrent Neural Networks. *CoRR*, abs/1901.04555, 2019.

[81] Eleanor Selfridge-Field, editor. *Beyond MIDI: The Handbook of Musical Codes*. MIT Press, Cambridge, MA, USA, 1997.

List of Figures

List of Tables

A. Attachments

A.1 Code

The code with programs for similarity analysis, form extraction, data preparation, neural network models, clustering analysis and programs for miscellaneous experiments and picture creations are in folder *code*.

A.2 Documentation

The documentation which describes how to use all the attached programs and comments their design is in *user_guide/user_doc.pdf*. The documentation also includes citations for all of the used libraries and subprograms.

A.3 List of Data

A list with names of the midifiles from `www.kunstderfuge.com`, that were used for training in this work is provided in file *dataset_list.txt* The data is attached in *dataset*, but is encrypted with password on demand due to possible copyright claims.

A.4 Dictionary of Instruments

Instruments
Violin, Viola, Cello, Contrabass, Higher Sax, Tremolo strings,
Pizzicato Strings, Harp, Undefined,String ensemble,
Slow strings, Choir, Voice, Orchestra hit,
Trumpet, Trombone, French horn, Higher Sax,
Baritone Sax, Oboe, Bassoon, Clarinet, Flute,
Bagpipe,Tinkle Bell, Steel Drums, Reverse Cymbal,
Instrument Families
Piano, Chromatic Percussion, Organ, Guitar,
Bass, Strings, Ensemble, Brass, Reed, Pipe, Synth Lead,
Synth Pad, Synth Effects, Ethinc, Percussive, Sound Effects

Table A.1: If the specific MIDI instrument key is found in the Instruments part of the table, we view it as a specification of an instrument, if the key is not found, we view it as the class of the MIDI instrument family as defined by the MIDI [81] technical standard. The *Undefined* instrument stands for percussion-like instruments that play a special role in compositions

A.5 Clustering Analysis

All the clustering analysis are in *cluster_analysis/interesting_measures*. The files for the analysis are stored as *MIDI* files in folders indicating their "parent" composition, relation i.e. intra-closest, inter-closest and intra-farthest. The name of the *midi* file itself indicates the position within a composition as "*a_b*", where *a* is the track index and *b* is the measure number.

As storing them as *png* files is space demanding, we provide the algorithm to create pictures from these midi files for individual compositions of interest.

Midis with inscribed forms are in *cluster_analysis/labeled_midis*. Figure 6.8 is in *cluster_analysis/labeled_midis/piano/dvorak/major/ dvorak_concerto_2-pianos_2_(c)yogore.mid.with_text.mid*.

A.6 Categories of Time Signatures

Time Signatures
4/4, 3/4, 2/4, 6/4, 9/4, 12/4, 2/2, 3/8,
6/8, 9/8, 12/8, 5/4, 3/2, 4/2, 7/16, 4/8,
1/8, 2/8, 1/4, 1/16, 9/16, 7/8, 5/8, 7/4,
1/32, 8/8, 1/2, 11/8, 13/16, 8/4, 17/32, 5/32,
6/8, 6/32, 3/16, 5/16, 11/16, 10/8, 2/16, 10/4,

Table A.2: Table of categories of the time signatures encountered and used for conditioning the models. Each of the category is transformed to an embedding which is concatenated to the input vector. The use of this information is for the model to know the beat structure and total duration of the desired output

A.7 Musical Forms Inscribed into the Data

All the musical forms are in *cluster_analysis/labeled_midis*.

To properly view these forms, it is recommended to use Musescore [64] software, while setting "Split Staff" to false and possibly changing how Musescore displays the lyrics using *Style* menu.

Figure 6.7 is in *cluster_analysis/labeled_midis/haydn/major/ haydn_catchi_hob_xxxic16_2_(c)iscenko.mid_with_text.mid*

A.8 Generated Examples

The generated examples are stored as midi files in folder *generated_examples*.

- The examples of the output from the variants generator are in subfolder *variants_compositions*.
- The examples of the generated compositions are in subfolder *reproducible_samples*.

- The examples of the generated symphonic compositions are in subfolder *reproducible_samples_symphonic*
- The examples of the generated long (200 measures) compositions are in subfolder *reproducible_samples_with_extra_long*.
- The examples for FID instrumenation comparison tables 9.3, 9.2 are in subfolder *reproducible_samples_piano*, and *reproducible_samples_string_quartet* and the FID results Table 9.1 in examples in subfolder *evaluation_generic*.