Department of Computer Science and Engineering, The Chinese University of Hong Kong

Final Year Project - KY1701

Final Report
Automatic Piano Reduction (Backend): Chord Identification

By JIN Xiamu 1155046896
with teammate Li Cheuk Yau 1155048011

Supervisor: Prof. Kevin Yip
Co-supervisor: Prof. Lucas Wong

Table of Content

1. Introduction

1.1 Background

Music is always considered as one of the greatest arts in the world, it provides both recreational and spiritual comforts to human beings. Throughout the history, thousands of millions of musical masterpieces are created by musicians. However, one cannot always find all parts or all instruments that are used by the composer originally, especially when it comes to orchestral arrangement, where dozens of uncommon instruments used in a whole music piece. Therefore, performing reduction on complicated music pieces becomes a crucial topic.

However, reduction requires a lot of musical and mathematical analysis in the original score. Different musicians may hold different opinions on how the reduction should be processed. Thus, reduction is not a simple and unidimensional task that is easy to be dealt with.

Yet recently, machine learning technology has been improved and developed quickly, it becomes to be applied to almost every aspects in our daily life. Reduction using computational algorithms by learning musicians' reduction pattern becomes realistic. Obviously, reduction performed by machine learning method would be much less time-consuming compared with traditional human calculation, and, if optimal algorithms are applied, it would be even more precise than traditional methods.

The goal of the whole project is to develop an automatic piano reduction software tool. This year, our group focus on chord identifications. We continued on the works of the previous years, collaborating with the front-end team and machine learning team on developing a chord identification application. The main focus of our team is to improve the accuracy and develop a more general reduction method that can suit more types of pieces.

1.2 Piano Reduction

A piano reduction is an arrangement in which sheet music for the piano that has been compressed so as to fit on a two-line staff and be playable on the piano. Piano reduction is a subjective process since different musicians will have their own recognitions on how harmony changes. Thus questions have been raised: which notes should be kept and which should be removed? After the reduction, is the score still playable without destroying the harmony?

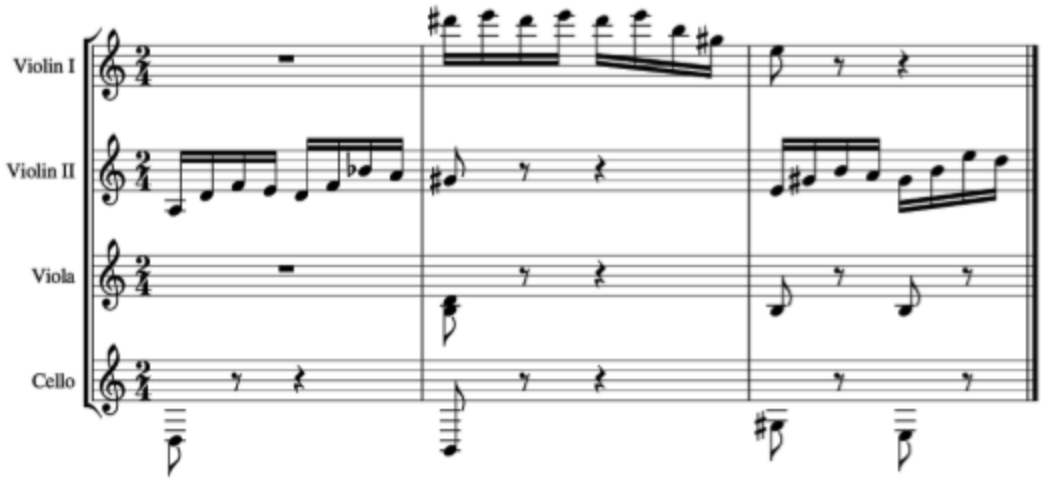


Figure 1.2.1 Excerpt of String Quartet (SQ-Original.xml)

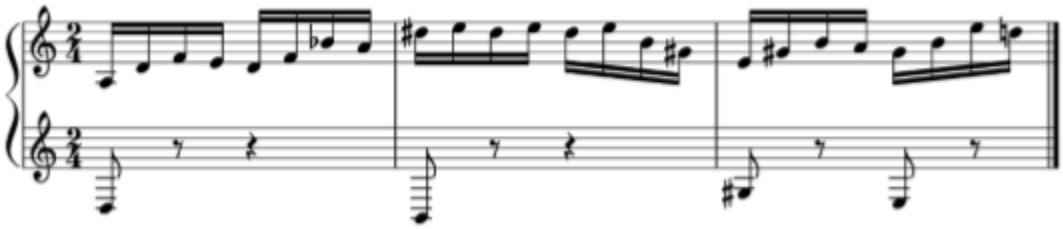


Figure 1.2.2 Excerpt of piano-reduced String Quartet (SQ-Important entrances plus bass line 2(more playable).xml)

1.3 Objectives

The main goals for us in this semester are as followed:

1. Design a Hidden Markov Model for chord transition analysis
2. Apply a rule-based algorithm for chord analysis -- ASHES
3. Present results by embedding labels in output MusicXML files

1.4 Music Theory

1.4.1 Note and Accidental

In music, {C, D, E, F, G, A, B} are used to represent pitches while an octave number in range [-1, 9] may be added behind to distinct two pitches having the same note name. For instance, C4(~ 261 Hz) is the ‘middle C’ on the music keyboard and C5(~ 523 Hz) is another note named C which is an octave higher than C4.

Apart from the seven note used, there are totally five accidental symbols which are used to alter the note by 'semitone', or in other name, half step. A semitone or half step has a frequency ratio of $\sqrt[12]{2}$, approximately 1.059.

The five accidentals are: 1. Sharp ♯, raises a note by a semitone or half-step, and 2. Flat ♭, lowers it by the same amount. 3. Double-sharp 𝄪, raising the frequency by two semitones, and 4. Double-flat 𝄫, lowering it by two semitones. 5. A special accidental, the natural symbol ♮, is used to indicate an unmodified pitch.

1	2	3	4	5	6
C	C sharp (C♯)	D	D sharp (D♯)	E	F
	D flat (D♭)		E flat (E♭)		

7	8	9	10	11	12
F sharp (F♯)	G	G sharp (G♯)	A	A sharp (A♯)	B
G flat (G♭)		A flat (A♭)		B flat (B♭)	

Figure 1.4.1.1 Naming Convention of 12-tone Chromatic Scale Built on C

1.4.2 Chord

Two or more notes sounded simultaneously are known as a chord. (Ottó, 1991) Chords are formed by adding two or more distinct frequency to produce harmonic effects, which vivid the expressiveness of music. Therefore, understanding the chords of piece is essential to analysis a music score and undergo reduction process.

In chords, notes may often given other names in while discussing chords. For instance, the fundamental note that chords are built on would name as Root, while other notes would name as second, third, so on and so forth depending the note difference from the root. If C major chords are being studied, the second would be D, the third would be E.

1.4.2.1 Chord Romanization

Roman numeral system bestows chords the "meaning" by using roman number (I,II,III,IV). It is also referred as scale degree in music. The key idea is to denotes a series of chords that are

related to a particular music key. It is important because if we want to have a better picture of a music score, their music key must be known.

The table below shows a simplified roman numeral table in C major

I (major triad)	C E G (C major)
II (minor triad)	D A F (D minor)
III (minor triad)	E G B (E minor)
IV (major triad)	F A C (F major)
V (major triad)	G B D (G major)
VI (minor triad)	A C E (A minor)
VII (diminished triad)	B D F (B diminished)

1.4.2.2 Chord Inversion

A chord inversion describes the relationship of its bass to the other tones in the chord. For instance, a C major triad contains the tones C, E and G; its inversion is determined by which of these tones is the bottom note in the chord. C major chord is composed of C,E,G. If E becomes the bass note (E,G,C), the chord is in first inversion. If G becomes the bass note (G,C,E), the chord is in second inversion.

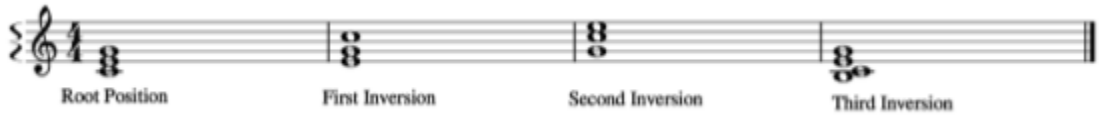


Figure 1.4.2.2 The root position and three inversions of C major 7th chord

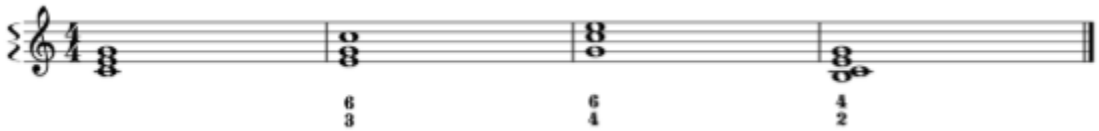


Figure 1.4.2.3 The figured bass notation of inversions

1.4.2.3 Chord Voicing

In music theory, chord voicing is the manner in which one distributes notes and chords among the various instruments. It includes the instrumentation and vertical spacing and ordering of the

musical notes in a chord: which notes are on the top or in the middle, which ones are doubled, which octave each is in, and which instruments or voices perform each note.

Typically, voicing has two position: Close Position and Open Position. For instance, C major chord, which is composed of C, E, G, is played as C4E4G4 in close position. In this case, all notes are played on the same octave. While in open position, C major chord can be played as C4E5G4, C5E4E6 or other different octave arrangements.

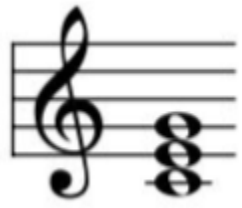


Figure 1.4.2.3.1
C major chord, Close Position

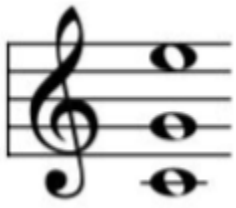


Figure 1.4.2.3.2
C major chord. Open Position

1.4.3 Beats

1.4.3.1. Beat

In music and music theory, beat is literally the basic unit of time, the pulse, of the measural level. By figuring out the beat of a certain music piece, we are not only able to understand the time attribute of the piece, we can also get clues of its rhythm pattern and chord pattern.

1.4.3.2. Time Signature

In music, a time signature tells you the meter of the piece you're playing. Composers decide the number of beats per measure early on and convey this information with a time signature. It is formed by two numbers -- the upper one denotes the number of beats and the lower one defines the unit of each beat. For example, a 4/4 time signature indicates that each measure contains four quarter note beats.

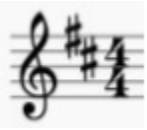


Figure 1.4.3.2 A 4/4 Time Signature

1.4.4 Keys

In music theory, the key of a piece is the group of pitches, or scale that form the basis of a music composition in classical, Western art, and Western pop music. A key specify a set of notes that follow a specific pattern, which would be commonly used in the piece. The keys could be major or minor, both of which follow a similar pattern, containing a key signature and seven notes.

1.4.4.1. Key Signature

In musical notation, a key signature is a set of sharp (♯), flat (♭), and rarely, natural (♮) symbols placed together on the staff. It designates notes that are to be played higher or lower than the corresponding natural notes and applies through to the end of the piece or up to the next key signature. For example, the key signature of A major/ F# minor has three sharps, F#, C# and G#, indicating that every F, C and G has to be modified one semitone higher.

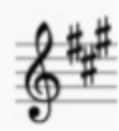


Figure 1.4.4.1 A major/ F# minor Key Signature

1.4.4.2. Major and Minor

Major and minor are the two most common key types used in traditional western music. Both key types contain 7 notes in an octave but having different intervals between notes.

We shall focus on the differences between the major key and the minor key by illustrating C major and C minor in the remaining of this section.

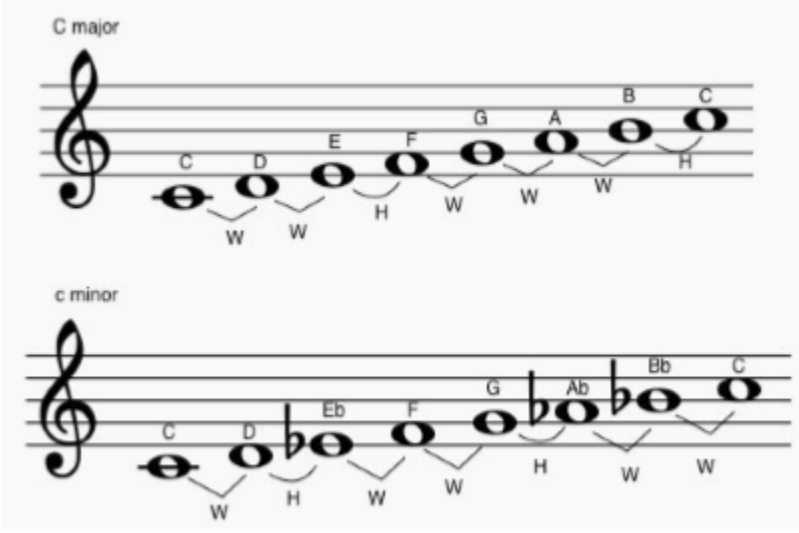


Figure 1.4.4.2 C major and C minor

The above figure shows the scales, which ordered the notes included in the specific key in increasing pitch. The 'W' and 'H' indicates the intervals between the two notes where 'W' stands for a full step (2 semitones) and 'H' stands for a half step (1 semitones). In any scale of major and minor, there exists 5 full steps and 2 half steps with position same as the above figure.

1.4.5. Melodic Decoration

1.4.5.1. Passing Notes

A passing note falls in between two different notes a third apart. For example, as the figure below has shown, the notes C and E are a third apart. The D falls between them, so it is a passing note.

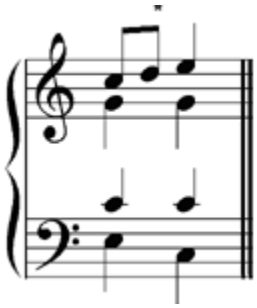


Figure 1.4.5.1 Passing Notes (the D between C and E)

1.4.5.2. Auxiliary Notes (also called "Neighbour Notes")

An auxiliary note falls between two identical chord notes. It can be higher or lower than the chord note. An auxiliary note which is higher than the chord note is an "upper auxiliary note" and a "lower auxiliary note" is lower than the chord note. Also, auxiliary notes can be either accented or unaccented, just like passing notes.

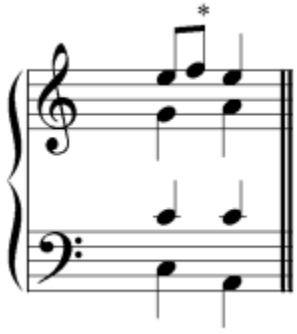


Figure 1.4.5.2 Auxiliary Note (marked with *)

1.4.5.3. Anticipation

An anticipation happens when we write one chord note earlier than the rest of the chord - in the beat before the rest of the chord sounds. Here, the B is part of the G major chord. The G major chord is sounded on the 2nd beat, but the B is sounded earlier, on the half beat before, so it is

an anticipation. Anticipations are usually approached by a downwards motion (e.g the C falls to B).

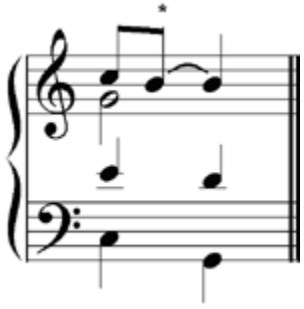


Figure 1.4.5.3 Anticipation

1.4.5.4. Suspension

Suspensions are the opposite of anticipations.

A suspension happens when we write one chord note later than the rest of the chord - during the beat after the rest of the chord sounds. In this example, the B doesn't sound immediately with the rest of the G major chord - instead, the C from the C major chord is held on for a little longer, and then falls to the B half a beat after the G major chord has sounded. The C is not part of the G major chord, so it is a non-chord note. The C is a suspension.

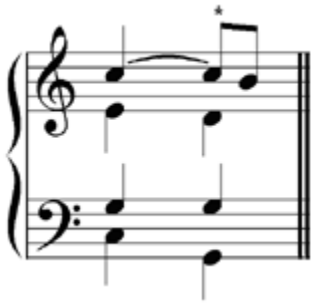


Figure 1.4.5.4 Suspension

1.4.5.5. Pedal

A pedal is either the tonic or dominant note played in one part continuously, while the chords in the other voices change.

Pedals normally occur in the bass, (but it is possible to find them in any of the other voices too). The pedal note is either held on for a long time, or repeated several times.

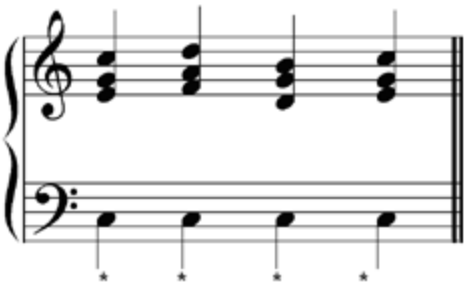


Figure 1.4.5.5.1 A Tonic Pedal

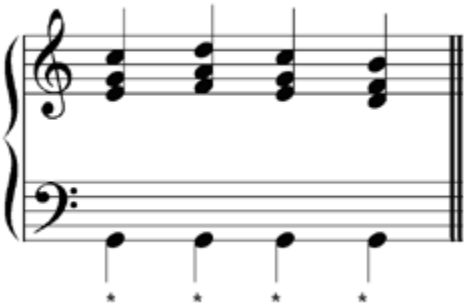


Figure 1.4.5.5.2 A Dominant Pedal

1.4.6. Music Score

Music score is handwritten, printed or digitalized form of music notation that users can indicate pitches, rhythm and other basic elements to allow them to play on instruments.

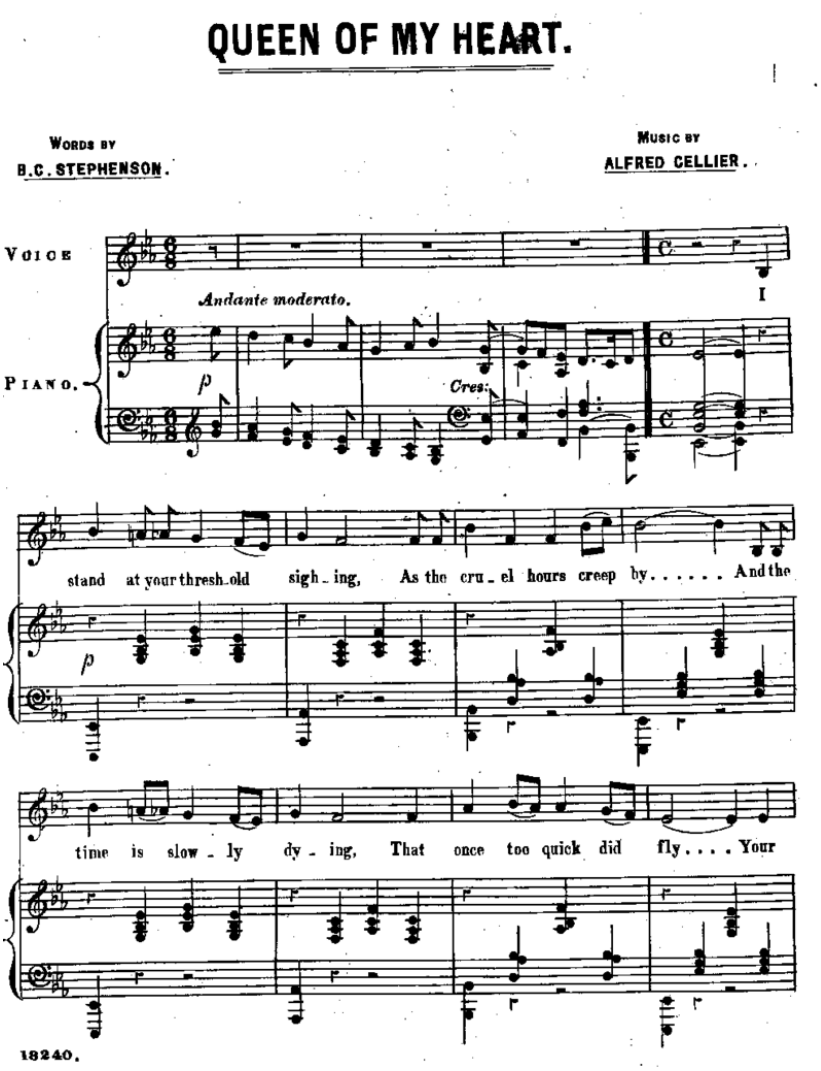


Figure 1.4.6 (https://en.wikipedia.org/wiki/Sheet_music#/media/File:QoMH.png)

1.5. Technical Support and Tools Used

1. Python 3.6: main computer language for programming.
2. music21: a toolkit for computer-aided musicology. It is used for parsing and retrieving music information of files in MusicXML format.
3. MusicXML: a standard format for representing digital sheet music.
4. MuseScore: open source music composition and notation software for viewing and editing music files, including MusicXML.
5. hmmlearn: a open source toolkit to learn Hidden Markov Model on python

2. Previous works

2.1 Works by Us from Previous Semester

In the previous semester, we focused on developing an algorithmic way to perform chord identification. However, since we targeted on developing a Hidden Markov Model for chord transition analysis in this semester, not much modules are reused.

2.1.1 Preprocessing for musicXML scores

Lines for different instruments can be written in the same staff by two voices or chords. In order to handle cases of chords. We use a built-in function Stream.voicesToParts() in music21 library to split second voices to a new part.

To facilitate slicing measures in to beat segments, ***Stream.sliceByBeat()*** is introduced for the whole score, which slice all elements in the Stream that have a Duration at the offsets determined to be the beat from the local ***TimeSignature***. Notes will be changed to tied notes if their duration is longer than a beat length.

For the work in this semester, we reuse the preprocessing method in the main, for both HMM analysis and ASHES (refer to section 4.3 and 5.3.1 for more details).

3. Overall System

Based on what we have achieved in the last semester, we found that analyzing probability on chord progression might be a feasible way to help with chord identification. Therefore, we decided to mainly focus on developing a Hidden Markov Model for chord transition analysis in this semester, where notes would be used as input data and chords would be presented as hidden states. Meanwhile, we also developed a rule-based algorithm, namely, ASHES, which refers to an unpublished paper by Professor Lucas Wong. We applied these two methods on the data that we have collected, comparing the results and then figuring out what could possibly be improved.

3.1. Individual Components

Both of the two methods are consisted of several individual components, each of which provides indispensable function in the whole system. A brief introduction of the usage of these components will be illustrated below.

3.1.1 Hidden Markov Model

By analyzing the probability of chord transitions, the result of chord identification would be more promising. Therefore, we decided to use Hidden Markov Model to learn the probability of transition between certain chords. It is composed of three parts, namely, ***Preprocessor***, ***hmmlearn*** and ***Postprocessor***. The details of this Hidden Markov Model including model design, mechanism and implementation will be illustrated in Section 4.

3.1.1.1 Preprocessor

Preprocessor is the component which is in charge of converting a MusicXML file into our desired input data format. The program will take a whole MusicXML file as input, identifying the occurences of the 12-pitches (see Section 4.2.1) in each time unit and converting them into a format of a list of vectors. Detailed mechanism are illustrated in Section 4.3.

3.1.1.2 hmmlearn

hmmlearn is the main part of implementing our Hidden Markov Model. It takes preprocessed input data, i.e. 12-dimensional vectors and their corresponding chords as input, and generates a chord transition matrix illustrating the probability of certain chords transiting into other chords. Detailed mechanism are illustrated in Section 4.4.

3.1.1.3 Postprocessor

Postprocessor is used to meet our third objective, that is to present results by embedding labels in output MusicXML files. By applying chord transition matrix on tested music scores, **hmmlearn** would generate a list of predicted chords for each time unit. This program could

visualize the result on corresponding MusicXML files. Detailed mechanism are illustrated in Section 4.5.

3.1.2 ASHES

3.1.2.1 Preprocessing

Before implementing the matching process, ***preprocessing*** is done to convert the musicXML files into a 2d grid array of music.notes for each measures, fill rests and other preliminary calculations. Refer to Section 5.3.1.1 for more details.

3.1.2.2 Vertical Match

This process is responsible for finding every possible perfectly matched triads or partially matched triads from each beat unit, by using the pitch counter statistics that was prepared in the preprocessing stage. Detailed mechanism are illustrated in 5.3.1.3

3.1.2.3 Vertical Merge

The intermediate triad results from ***Vertical Match*** will be merged to 7^{th} chord if possible in this stage. Best matching triad selection will be performed if more than one perfect matching triads exist for each beat unit. Detailed mechanism are illustrated in 5.3.1.4

3.1.2.4 Horizontal Merge

Based on the ***Vertical merging*** results from the previous stage, further elimination will be done here. ***Horizontal Merge*** is responsible for maximising the chord selection results by checking for possible merges between beat units, with the assistance of the cut-off beat found in the ***preprocessing*** stage. Detailed mechanism are illustrated in 5.3.1.5

3.2. Relational Graph

Please refer to Section 4.6 and 5.2.2 for detailed relational graph of HMM and ASHES.

4. Hidden Markov Model

4.1 Basic Introduction to Hidden Markov Model

4.1.1 First Order Markov Chain

A random variable X which takes discrete values from a state space Ωx of size N, $X \in \Omega x = \{1,\ldots,N\}$. The series $X = (x1, x2, \ldots, xT)$ over the time steps $t = 1,\ldots,T$ and for $xt \in \Omega x$.. The series is called a first order Markov chain if the conditional probability of the state at time step t given all the previous states, depends only on the last states. Hence, a first order Markov chain has the following property:

$$p(x_t|x_{t-1},\ldots,x_1) = p(x_t|x_{t-1})$$

The following graph is a representation of first order Markov chain, where the arrows show dependencies, illustrating the first order markov property, as a node only depends on the previous node.

Figure 4.1.1.1 Illustration of a First Order Markov Chain

In first order hidden Markov chain, the probability of transiting from one state i to another state j is always the same for every time step in the chain and we denote it by P_{ij}.

$$P_{ij} = p(x_t = j|x_{t-1} = i)$$

The transition probability is often denoted by a N*N transition matrix with symbol P.

$$\boldsymbol{P} = \begin{pmatrix} P_{11} & P_{12} & \ldots & P_{1N} \\ P_{21} & P_{22} & \ldots & P_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ P_{N1} & P_{N2} & \ldots & P_{NN} \end{pmatrix}$$

Figure 4.1.1.2 Transition Probability Matrix

For a first order markov chain, the mass function for the whole sequence is given by:

$p(X) = p(x_1) \cdot \prod_{t=2}^{T} p(x_t|x_{t-1})$, where $p(x_1)$can be defined from an initial distribution, namely, priors, and the values of $(x_t|x_{t-1})$can be found in the transition matrix P.

4.1.2 Hidden Markov Model

Suppose that we have a first order markov chain, $X = (x_1, \ldots, x_T)$. In each time step t, an output could be observed, and we denote it by y_t. Thus we could get a series of observed output $Y = (y_1, \ldots, y_T)$ The values of observed output Y are conditionally independent to X, however, each individual value y_t is dependent a single site of$X$, x_t. In other words, the observation y_t depends only on the variable x_t. Thus we have the following likelihood distribution:

$$p(X|Y) = \prod_{t=1}^{T} p(y_t|X) = \prod_{t=1}^{T} p(y_t|x_t)$$

When only Y is observed and the underlying variables X are said to be unknown or “hidden”, an HMM would be defined. The following graph is an illustration of first order HMM, where the top green level represents the observed output Y and the bottom blue level represents the hidden states, which is also the same as the first order Markov chain shown in Figure 4.1.1.1.

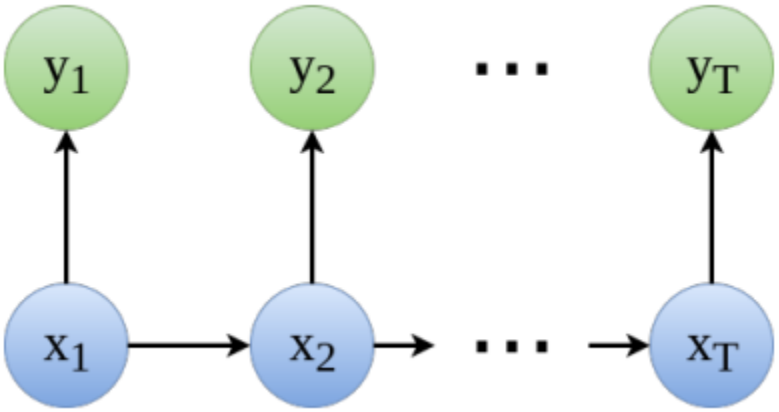


Figure 4.1.2 First Order HMM

4.1.3 Viterbi Algorithm

The Viterbi Algorithm is an computational method which is used to find the sequence of hidden states that is most likely to appear in a HMM. The pseudo codes are as followed:

Input:

- The observation space $O = \{o_1, o_2, \ldots, o_N\}$
- The state space $S = \{s_1, s_2, \ldots, s_K\}$
- An array of initial probabilities $\Pi = (\pi_1, \pi_2, \ldots, \pi_K)$such that π_i stores the probability of $x_1 == s_i$
- A sequence of observations $Y = (y_1, y_2, \ldots, y_T)$such that $y_t == i$ if the observation at time t is o_i
- Transition matrix A of size K by K
- Emission matrix B of size K by N

```
function VITERBI(□,□,□,□,□): X
    for each state □ ∈ {1,2,...,□} do
        □1[□, 1]← π□ · □□□1
        □2[□, 1]← 0
    end for
    for each observation □ = 2,3,...,□ do
        for each state □ ∈ {1,2,...,□} do
            □1[□,□]← □□□□(□1[□,□ − 1] · □□□ · □□□□)
            □2[□,□]← □□□□□□□(□1[□,□ − 1] · □□□ · □□□□)
        end for
    end for
    □□← □□□□□□□□□1[□,□]
    □□←□□□
    for □←□,□ − 1,...,2 do
        □□−1← □2[□□,□]
        □□−1←□□□−1
    end for
    return X
end function
```

4.2 Model Design

A certain musical score with chords can be represented in a HMM model by making x_t represent the chord at time step t, while letting y_t represent the musical notes which appears in the corresponding time step. In this way, we could conclude that the notes in the score are considered to be observed output and chords are regarded as hidden states.

4.2.1 Input Format

In order to convert musical notes into datas that could be easily processed and calculated by computers, we need to define a proper input format that could elaborately illustrate musical

notes distributions in a certain time unit. We finally came up with a concise model that could be easily illustrated and understood by the computer.

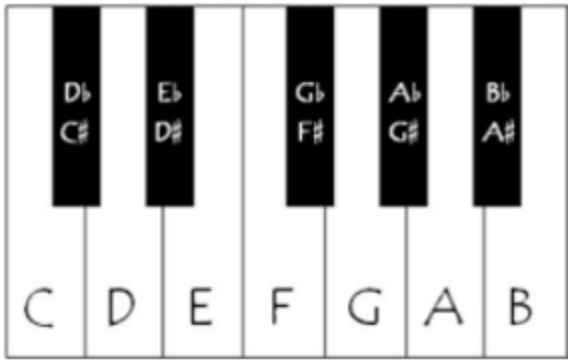


Figure 4.2.1.1 12 Pitches in an Octave

In our model, we take **get time unit from cutoff beats** (See Section 4.3.2). In a certain time unit, we collect all informations of its musical notes, and convert them into a **12-dimensional vector**. In music theory, a single octave, or chromatic scale, is consisted of 12 pitches, where two adjacent pitches have a difference of one semitone. Since we only need to identify the actual pitch of each notes when identifying chords while the octave does not necessarily matter, we could just take a look in the time unit, and see if a certain pitch exists in the time unit, then we assign 1 to that dimension, otherwise we will assign 0. The following table shows how pitches and dimensions are correlated.

Note	C	C#/ Db	D	D#/ Eb	E	F	F#/ Gb	G	G#/ Ab	A	A#/ Bb	B
Dimension	1	2	3	4	5	6	7	8	9	10	11	12

For example, the following part of a musical score is a measure consisted of seven notes: A, G, G, E, D, E, C. Thus in our model, it would be converted to a 12-dimensional vector:

$$y_t = [1,0,1,0,0,0,0,1,0,0,0,1]$$

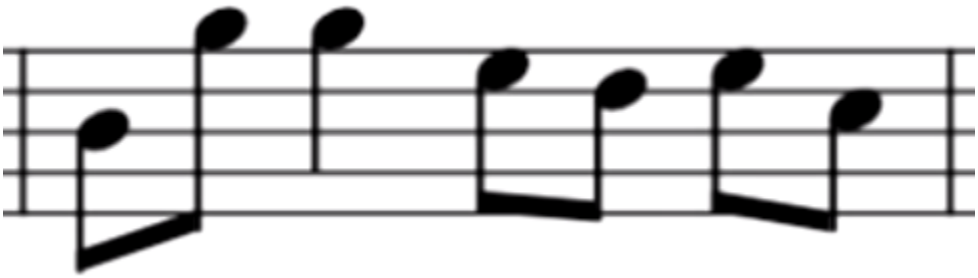


Figure 4.2.1.2 A measure that could be represented by $y_t = [1,0,1,0,0,0,0,1,0,0,0,1]$

4.2.2 Chords as Hidden States

In our model, chords are represented as hidden states corresponding to a certain measure of musical notes. The following table is a list of chords which Professor Lucas Wong provided for us for reference.

Keys	C, D, E, F, G, A, B C#, D#, E#, F#, G#, A#, B# C##, D##, E##, F##, G##, A##, B## Cb, Db, Eb, Fb, Gb, Ab, Bb Cbb, Dbb, Ebb, Fbb, Gbb, Abb, Bbb
Chords in major scale	I, I 7, II b, II, II 7, III, III 7, IV, IV 7, V, V 7, VI b, VI German, VI French, VI Italian, VI, VI 7, VII, VII 7, VII dimnish7
Chords in minor scale	I, I +, II b, II, II 7, III, IV, IV +, V, V +, V +7, VI b, VI German, VI French, VI Italian, VII, VII dimnish, VII dimnish7

In our model, key change and chord transition are treated separately, which means that we have prepared different models, i.e. chord transition matrix, for different keys. Thus keys should be known and labeled prior to the training part. Also, in order to simplify the chord transition table, we have merged major chords and minor chords thus yield the following chord list:

I, I +, I 7, II b, II, II 7, III, III 7, IV, IV +, IV 7, V, V+, V 7, V +7, VI b, VI German, VI French, VI Italian, VI, VI 7, VII, VII 7, VII diminish VII dimnish7

4.2.3 Graph Illustration

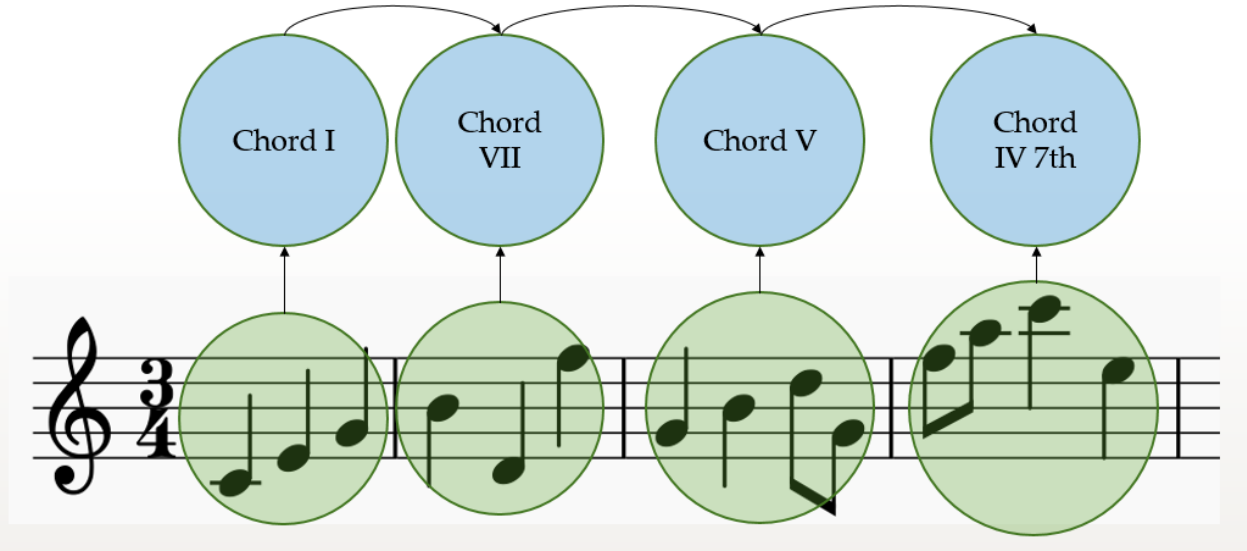


Figure 4.2.3 Graph Illustration of Our Model on C major

The above graph is an illustration of our model on C major. The state x_t are represented in blue circles while the observed notes are represented in green ones. The length of each time step corresponds to the length of a measure in the musical score. The arrows represent the dependencies in the model.

4.2.4 Emission Probability

In Hidden Markov Models, emission probabilities are the conditional distributions of the observed variables p(xn | zn) from a specific state. In our model, the emission probabilities are set according to some rules we defined.

We define a term called ‘degree of similarity’, which describes the similarity of two input vectors. After consulting Lucas about musical theory, we know the fact that for different notes on a certain chord, the probability of the lowest note appearing is the highest, while the second lowest note and the highest note has relatively low probabilities to show up. For example, C Major chord I is consisted of three notes, C, E and G. Then if we generate a random note on C Major chord I, C has the highest probability to be generated thanks to the fact that it is the lowest note on the chord. Similarly, E has the second highest probability to be yielded, and G has the third highest probability. Other notes also have chances to get generated, but relatively low compared to those three notes.

Therefore, we define the emission probabilities for a certain input vector on a certain chord by the following rules.

1. When the lowest note is matched, +5
2. Second lowest note matched, +3
3. Third note matched, +2 (+1 for 7th chord)
4. 7th note matched, +1
5. The occurrence of other notes, -1 for each

According to the above rule, we have 11 levels of ‘degree of similarity’, from 0 to 10. How emission probabilities are calculated from degree of similarity are shown as below.

deg(sim) = 10 → 10/55 (e.g. C E G on C major chord I)
deg(sim) = 9 → Sharing the probability of 9/55 (e.g. C E G B on C major chord I)
deg(sim) = 8 → Sharing the probability of 8/55
deg(sim) = 7 → Sharing the probability of 7/55
deg(sim) = 6 → Sharing the probability of 6/55
deg(sim) = 5 → Sharing the probability of 5/55
deg(sim) = 4 → Sharing the probability of 4/55
deg(sim) = 3 → Sharing the probability of 3/55
deg(sim) = 2 → Sharing the probability of 2/55
deg(sim) = 1 → Sharing the probability of 1/55
deg(sim) = 0 → 0

4.3 Preprocessor

4.3.1 Rests substitution

In each measure and part that contains both rest(s) and note(s), the rest at the beginning of the measure would be replaced by the first pitch-class that appears in the measure. A note will sustain through the rest that occupies after it, until the next pitch-class or barline.

```
Procedure preprocessor(xmlFile)
      Strema = converter.parse(xmlFile)
      # rest filling
      For part in stream
            For n in part:
                  N = previous_n or next_n if n.isRest
      cutoff_beats = lowestMovingPart(measure).beats
```

4.3.2 Getting time unit from cutoff beat

To efficiently define a time unit for each to-be classified chord, we make use of the lowest moving part (i.e. part having lowest number of pitch-class movement) to dictate the cutoff beat. The lowest part movement is considered as an preliminary indicator for chord change.

```
Procedure lowestMovingPart(measure)
      For part in stream:
            For beat, n in part:
                  if n.pitchClass!= next_n.pitchClass
                        Cutoff[i].append(beat)
      lowestMovingPart= min(len(part_cutoff) for part_cutoff in cutoff)
      Return lowestMovingPart,cutoff_beats
```

4.3.3 Output as 12-dimensional vector

For each time unit, a **12-dimensional vector** is returned for HMM learning and testing. Enharmonic notes (notes with different name but same pitch in modern tuning) are treated as in the same dimension.

```
Procedure toHMMvector(measure, cutoff_beats)
      For beat in cutoff_beats:
            Counter = Counter([n.pitch%12 for n in
                  measure[beat:next_beat]])
            Vector = [counter[pitch] for pitch in range(12)]
            vectors.append((beat, vector))
      Return vectors
```

4.4 hmmlearn

hmmlearn is a open source python toolkit which implements Hidden Markov Model. The three fundamental problems of HMM, which are:

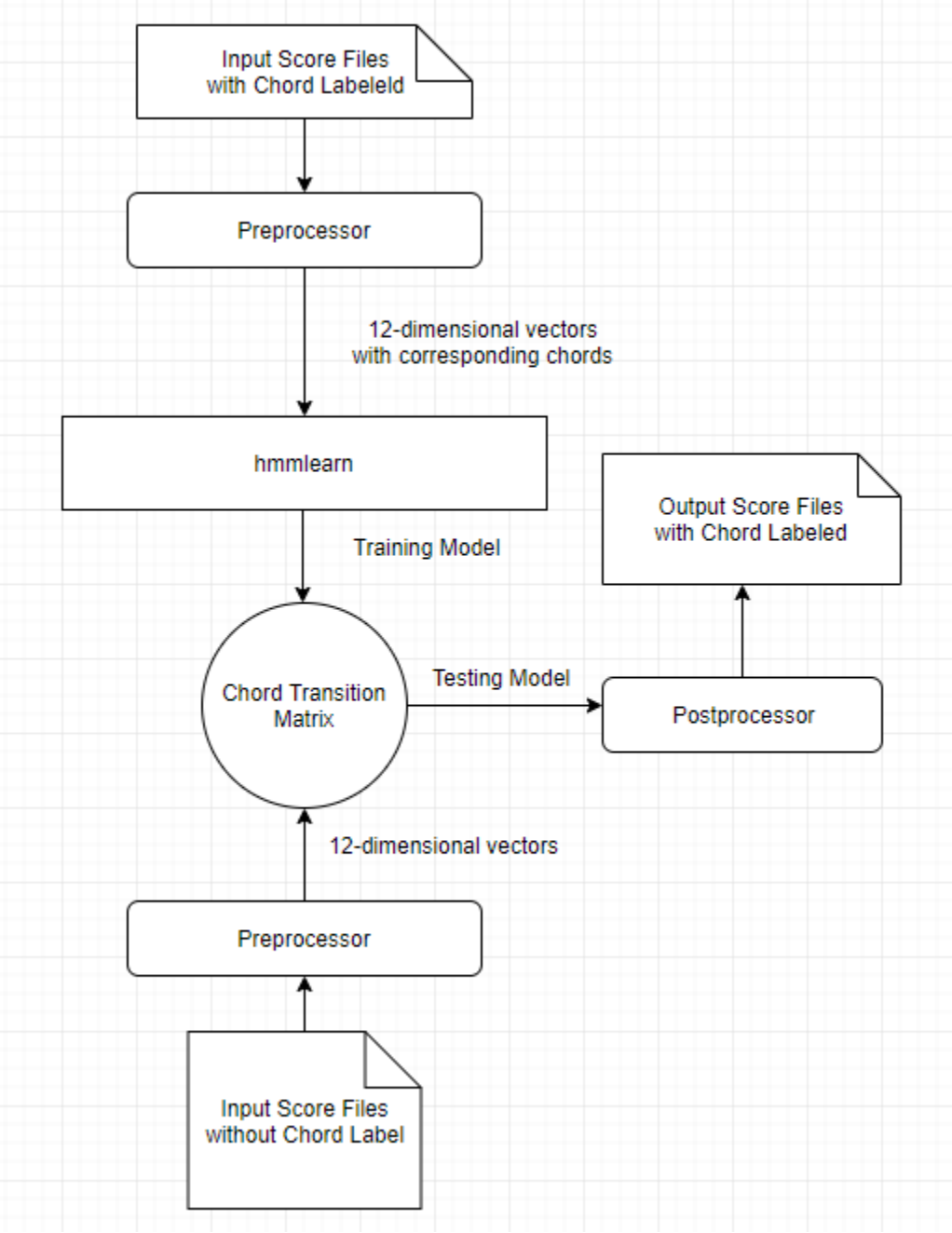
- Given the model parameters and observed data, estimate the optimal sequence of hidden states,
- Given the model parameters and observed data, calculate the likelihood of the data,
- Given just the observed data, estimate the model parameters,

could all be resolved by the built-in functions of **hmmlearn.** In our program, we mainly need to focus on the first (testing part) and the third (training part) problem.

4.5 Postprocessor

Postprocessor is simply done by inserting the selected chord label into the original musicXML files. The chord labels are added as the lyrics of the lowest note.

4.6 Overall Workflow Graph



4.7 Training Data Preparation

The training data collected by us includes three musical scores:

1. Beethoven Symphony No. 1 1st Movement (111 Chord Events, mainly C Major and G Major)
2. Beethoven Symphony No. 5 1st Movement (122 Chord Events, mainly C Minor, Eb Major and Bb Major)

3. Mozart Symphony No.25 The Little G minor Symphony - 1st Movement (214 Chord Events, G Minor)

4.8 Possible Improvement in the Future

Our HMM model serves as a tool to suggest chord analysis result to musicians. However, the correctness of our model is still not promising. There are a few possible improvements that could possibly be applied to our model, which we did not achieve, due to time and resource limitation.

4.8.1 Harmonic Analysis

The time unit of chord change is often not restricted to one measure, in fact, chord change does not occur at a fixed period of time, it could occur at anywhere. Thus it would be hard to define the basic time unit of Hidden Markov Model. Another challenge that could raise at the same time is how to resolve dissonances. Therefore, if an efficient harmonic analyzer could be applied and help our model define the right time unit, the accuracy of our model would be likely to get improved.

4.8.2 Collecting More Training Data

Due to the limitation of time and accessible resources, we did not find much labeled classic music scores to train our model. If more data could be collected, it is likely that, our Hidden Markov Model could have a better performance, or it would be easier for us to pinpoint the problem and fine tune the model.

5. Algorithmic Method - A.S.H.E.S.

5.1. Introduction

To compare the accuracy with hidden-markov model, we developed another algorithmic chord identification strategy with the assistance of Prof. Lucas Wong. Based on his experience as a professional musician, he generalised his chord matching strategy into ***Algorithmic Search for Harmonic Extraction and Simplification (ASHES)***.

5.2. Mechanism

5.2.1. Observations and Strategy

Harmony is mainly governed by chords, which can be decomposed into triads.
Here are the types of triads that we consider:

	Intervals between root & third	Interval between third & fifth
Major triad	Major 3rd	Minor 3rd
Minor triad	Minor 3rd	Major 3rd
Diminished triad	Minor 3rd	Minor 3rd
Augmented triad	Major 3rd	Major 3rd
Italian-6 triad	Diminished 3rd	Major 3rd

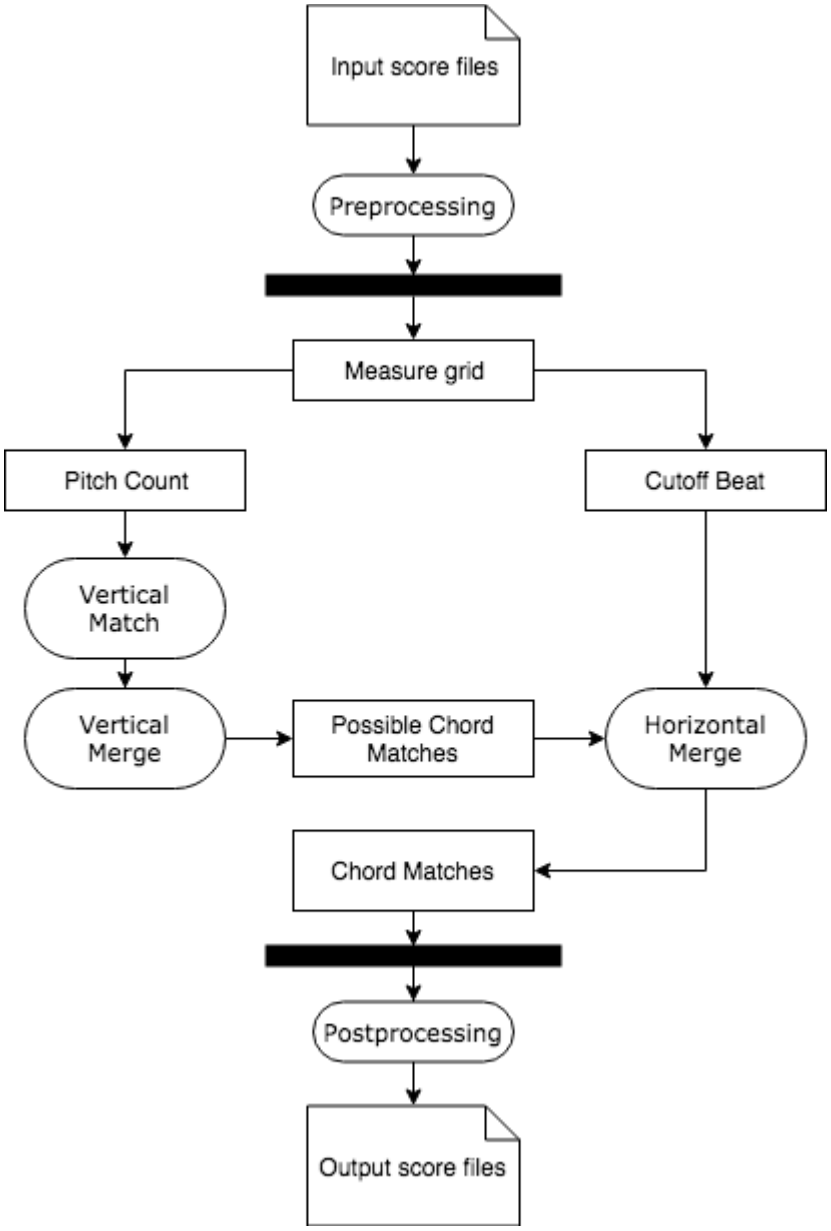
ASHES does not perform the chord matching by distinguishing the existence of dissonance notes, but instead search for the possible existence of triads in every time units, and perform 7th chord merging and selection afterwards.

	Triad formed by root, third and fifth	Triad formed by third, fifth and seventh
Major 7th	Major triad	Minor triad
Minor 7th	Minor triad	Major triad
Dominant 7th	Major triad	Diminished triad
Diminished 7th	Diminished triad	Diminished triad
Half-dimished 7th	Diminished triad	Minor triad

Western music has three types of textures: monody, homophony and polyphony. Here, we will consider monody as harmonically ambiguous and will not do a chord matching for them.

5.2.2. Programming workflow

ASHES mainly contains three steps. ***Vertical match***, ***vertical merge*** and ***horizontal merge***. In section 5.3, we will talk about the implementations in details.



5.3. Implementation

5.3.1. Class - Ashes

Ashes contains the main flow of the program.

5.3.1.1. Initialization

In *Initialization*, measure is preprocessed by converting musicXML files into music21 *Steam* class, removing grace notes, dividing voices in same staff to different staff and dividing each long notes wherever there are other parts changing. It perform the rest filling processing, similar to *Hidden Markov Model* ***Preprocessor*** (refer to section 4.3)
Measure_grid containing the notes of different parts at different beats, list of cutoff_beats and a list of pitch_count is initialized as the instance attributes.

```
stream = converter.parse(xmlName)
For measure in stream:
      initialization(measure)

Procedure initialization(measure)
      # preprocess
      self.measure = measure.removeGraceNotes().voiceToParts()
                  .getVoiceLeadingMoment()
      # instance attributes
      self.measure_grid = np.array(measure)       #grid with size
                                              len(parts)*len(beatCount)
      # rest filling
      For part in self.measure_grid
            For n in part:
                  N = previous_n or next_n if n.isRest
      self.Cutoff_beats = lowestMovingPart(measure).beats
      self.Pitch_count = Counter(for time_unit in measure_grid)
```

5.3.1.2. lowestMovingPart

This function is used in the initialization stage. The part with the slowest movement and the lowest pitch range will be returned, its changing beats will be further used as the primary cutoff beats. This process is also used in the *Hidden Markov Model* ***Preprocessor*** (refer to section 4.3)

```
Procedure lowestMovingPart
      For i in range(self.measure_grid.shape[0]):
            For j in range(self.measure_grid.shape[1]):
                  if measure_grid[i][j].pitch!=
                        measure_grid[i][j+1].pitch
                        Cutoff[i].append(beat)
```

```
    lowestMovingPart= min(len(part_cutoff) for part_cutoff in cutoff)
    Return lowestMovingPart,cutoff[i]
```

5.3.1.3. verticalMatch

For each vertical column in grid, every combinations of three from the occuring pitches will be grouped to perform triad check of perfect matches (all triad notes occur in column). If no perfect matches found, every combinations of 2 pitches will be grouped to perform partial matches.

```
Procedure verticalMatch
    For j in range(self.measure_grid.shape[1]:
        pitchCount = self.pitch_count[j]
        perfectMatch = [Triad.getPossibleTriad(nList) for nList in
            combinations(pitchCount, 3)]
        If not perfectMatch:
            partialMatch = [Traid.getPossibleTraid(nList) for
                nList in combinations(pitchCount, 2)]
    self.possibleTraids = perfectMatches or partialMatches
```

5.3.1.4. verticalMerge

For each vertical column in grid, check for any 7th chord matches for each possible matches found from previous stage. For cases of more than one perfect matches, do ***scoreTuple*** calculation (refer to 5.3.1.6) to select for the best match. For cases of only partial matches exist, keep all the candidates for next stage merging.

```
Procedure verticalMerge
    For j in range(self.measure_grid.shap[1]:
        possibleTraids = self.possibleTraids[j]
        Chord7th = Triad.mergeChord(combinations(possibleTraids,2))
        If (possibleTraids.getPerfect()+chord7th) > 1
            bestChord = max(scoreTuple(chord) for chord in
                possibleTraids)
    self.possibleChords = bestChord or possibleTraids
```

5.3.1.5. horizontalMerge

In ***preprocessing***, we generated a list of cutoff_beats which will be used as the primary cutoff in this stage.
Horizontal merging is performed twice. In stage 1, a chord candidate which have the highest existence will be chosen within each cutoff beat group. If no maximised chord is found, insert cutoff beat for where the maximised chord does exist anymore.
Stage 2 tries to perform a further merging within cutoff beats if common chords can be found.
If multiple matches still exist within cutoff beats, do scoreTuple calculation (refer to next session) again to select the best match.

```
Procedure horizontalMerge
      For j in range(self.measure_grid.shap[1]:
            bestChords = Traid.mergeChord(self.possibleChords[j],
                  bestChords)
            If not bestChords:
                  self.cutoff_beats.append(unmatch_beat)
                  stage1Chord.append(previous_bestChords)
                  bestChords = self.possibleChords[j]
            If beat[j+1] in self.cutoff_beat:
                  stage1Chord.append(bestChords)
      For beat in self.cutoff_beats:
            bestChord = Traid.mergeChord(
                  stage1Chord[beat],stage1Chord[next_beat]))
            If not bestChord:
                  Final_chords.append(previous_bestChord)
      Return final_chords
```

5.3.1.6. scoreTuple

ScoreTuple is used in both the ***verticalMerge*** and ***horizontalMerge*** stage when multiple chord candidates exists.
The three criteria scores are be calculated for *sorted()*:

withBass	if bassnote is in-chord, the chord has a higher priority
totalOccurance	The more instruments and duration playing the notes in chord, the higher the priority
missingPriority	Missing the 5th note is better than missing the 3rd note is better than missing the root

```
Procedure scoreTuple(chord, start_beat, end_beat)
      withBass = self.bassline[j_start] in chord
      totalOccurance = sum(self.pitchCount[start_beat:end_beat][pitch]
            for pitch in chord)
      missingPriority = chord.getmissingPriority()
      Return (withBass, totalOccurance, missingPriority)
```

5.3.2. Class - Triad

Triad is the class for all instances of chords (including 7th chords) found by ASHES.

5.3.2.1. getPossibleTriad

Given a pitch list of 2 to 3 notes, this function is called for generating any possible triads.

```
Procedure getPossibleTriad(nList)
      Intervals = [interval.Interval(n1,n2) for n1,n2 in
            nList[i],nList[i+1]]
      For triad in const.TRIAD:
            If intervals in triad.intervals
                  possibleTriads.append(triad.Triad(root=nList_root))
      Return possibleTriads
```

5.3.2.2. mergeChord

mergeChord has three main functions: The first one is merging the same chords and eliminating the missing notes at the same time (for ***scoreTuple*** calculation). Second is merging triads into 7th chord. Third is merging two triads into 7th if suitable.

```
Procedure mergeChord(c1, c1)
      If c1 == c2:
            c1.missingNote.remove(c2.existNote)
            Return c1
      If (c1.is7th and c2.notes in c1.notes) or
            (c2.is7th and c1.notes in c2.notes):
            C_7th.missingNote.remove(c_another.existNote)
            Return c_7th
      If isValid7th(c1,c2):
            C_7th = triad.Triad(c1.notes.union(c2.notes))
            Return c_7th
```

5.3.3 Postprocessing

Postprocessing is simply done by inserting the selected chord label into the original musicXML files. The chord labels are added as the lyrics of the lowest note.

5.4 Possible improvement in the Future

5.4.1 Dissonance forming triads

One major problem we found is when dissonance accidentally forming triads. This situation commonly occurs when multiple parts are playing running notes at the same time.

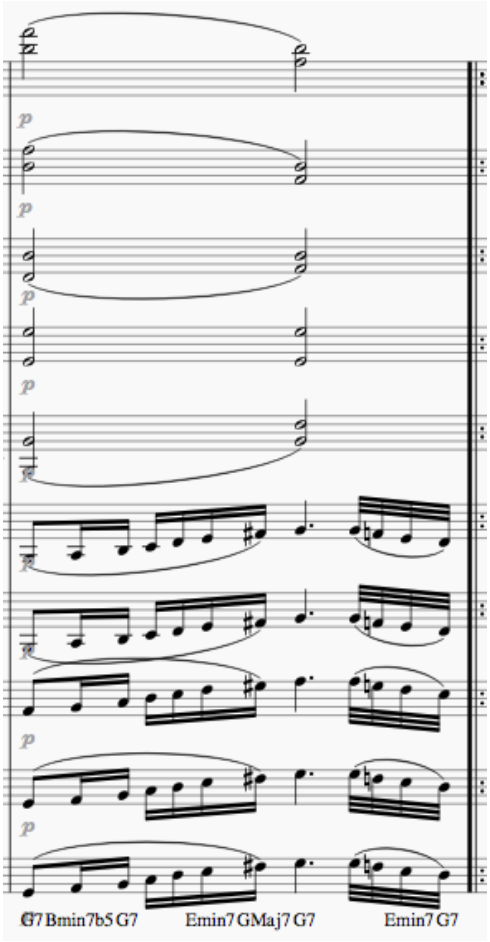


Figure 5.4.1 testing result extract

Here is a testing result extracted from Beethoven Symphony No.1 Mov.1 measure 12. The main harmony occurs on beat 1 and 3 which both have a perfect match of G7. However, neighbouring running notes forms triads of other chords. Postprocessing can be done to remove the unwanted notes not inside the main harmony progression.

6. Training and Testing

6.1 HMM Training Scores

Score	Chord Events	
Beethoven Symphony No. 1 1st Movement	111	mainly C Major and G Major
Beethoven Symphony No. 5 1st Movement	122	mainly C Minor, Eb Major and Bb Major
Mozart Symphony No.25 The Little G minor Symphony 1st Movement	214	G Minor

6.2 Testing Scores

Currently, we are testing the hidden Markov model and ASHES with the following scores:

Score	Measure number	Chord events (HMM)	Chord events (ASHES)
Pachelbel, Johann Canon in D	57	305	341
Mozart concerto K.314	11	30	30
Beethoven symphony OP.67	124(57 for HMM)	42	116
	Total events:	377	487

6.3 HMM Testing Result

6.3.1 List of Keys and Chords in the Training Data

List of Keys	Major Keys: C, D, E, Eb, F, G, Ab, Bb Minor Keys: C, D, F, G, A
List of Chords	I, I+, IIb, II, II7, III, IV, V, V7, V+, V+7, VIb, VI, VI7, VI Ger, VII, VII7, VII Ger, VII Ita, VII dim, VII dim7

6.3.2 Chord Transition Matrix Visualization

The following set of graphs are the visualization of chord transition matrices for each keys. Please note that the saturation of the color indicates the likelihood of the transition from two certain chords. The deeper the color is, the more likely that the transition would occur. The number on x and y axis indicates each chord, from 0 to 20, integers represent I, I+, IIb, II, II7, III, IV, V, V7, V+, V+7, VIb, VI, VI7, VI Ger, VII, VII7, VII Ger, VII Ita, VII dim, VII dim7, respectively.

G Minor:

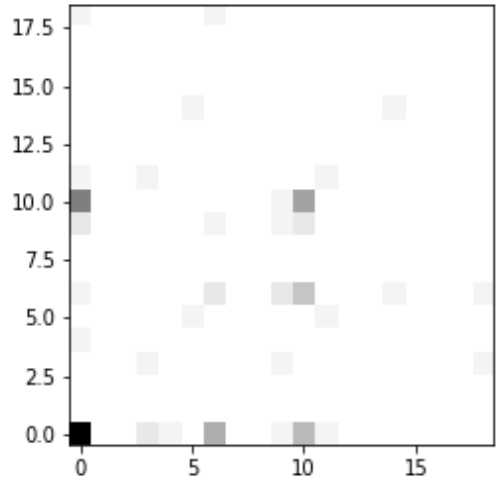


Figure 6.3.2.1 Transition Matrix for G Minor

C Minor:

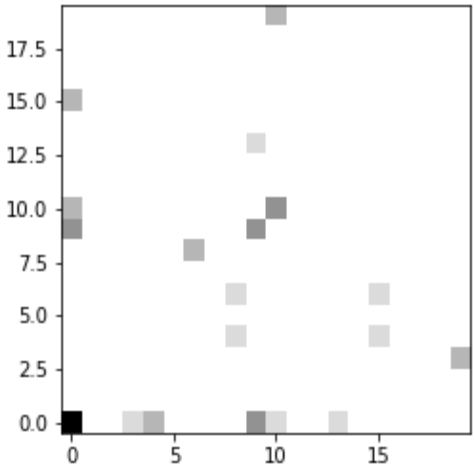
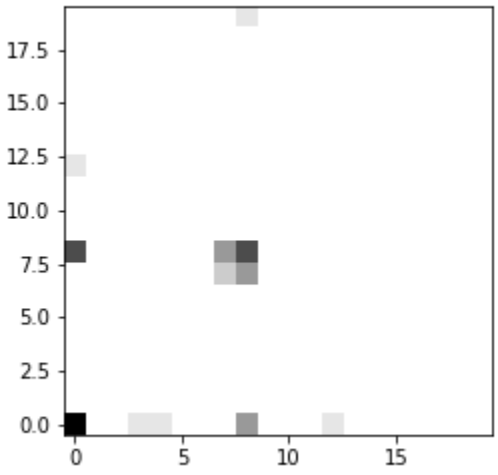


Figure 6.3.2.2 Transition Matrix for C Minor

C Major:



Eb Major:

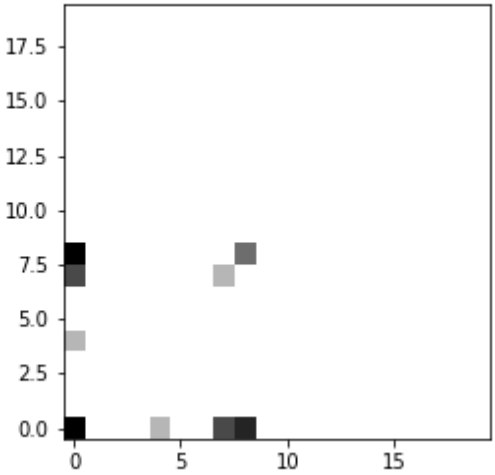


Figure 6.3.2.3 Transition Matrix for C Major

Figure 6.3.2.4 Transition Matrix for Eb Major

G Major:

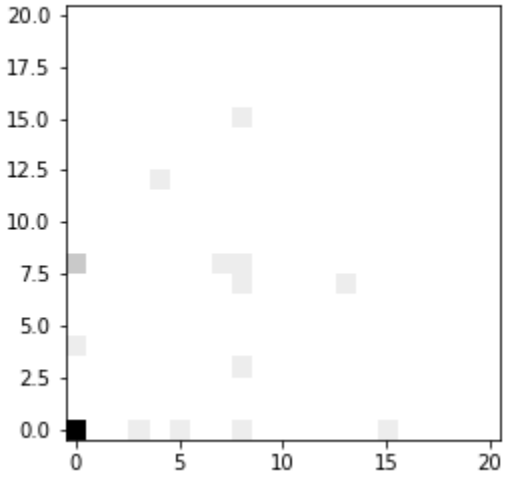


Figure 6.3.2.5 Transition Matrix for G Major

For the rest of keys appeared in the training data, due to the number of records being too low, the transition matrix visualization becomes meaningless thus would not be presented here.

6.4 Correctness Comparison

In the testing result analysis, we compared both the hard correctness and soft correctness. For soft correctness, overfitting chords are excluded in the calculation of correctness.

	ASHES	HMM

	Hard	*Soft*	Hard	*Soft*
Pachelbel, Johann Canon in D	65%	*89%*	56%	*78%*
Mozart concerto K.314	63%	*90%*	58%	*83%*
Beethoven symphony OP.67	80%	*83%*	64%	*78%*

Overall	**69%** (soft: 87%)	**58%** (soft: 80%)

The correctness of HMM results are mainly affected by two major factors: 1. Overfitting issue, where the causes are stated below as ASHES have the same problem. 2. Lacking of training data. Due to the limited time and accessible resources, we have only few records for transitions between certain chords. Thus, the probability of transition between certain two chords might be too high or too low (due to some certain type of chord transitions are missing). For example, Canon in D has a typical chord transition pattern of I-V-VI-III-IV-I-IV-V (repeat). However, our training data does not record any chord VI-III transition, thus the accuracy has been badly affected.

The correctness of ASHES results are mainly affected by overfitted chords, especially in the results of Canon in D and Mozart concerto K.314 (refer to section 5.4.1 for more details).

7. Conclusion

Overall, in this academic year, we have strived hard to came up with an efficient and accurate way to achieve our main objective -- chord identification. In the previous semester, we have designed a rule-based algorithm that achieved high accuracy in both scores we have tested. We used a hashing tree data structure and proper weighting score to help us identify the chord.

In order to improve the result, we mainly focus on constructing a chord transition probability table, which is exactly what Hidden Markov Model does. However, due to the difficulty on harmonic analysis, we could not find an optimal way to define the right time unit, plus the fact that we did not manage to get access to enough classical music scores with chord labeled, Hidden Markov Model does not yield in a promising result. A.S.H.E.S. is a rule-based algorithm proposed by Professor Lucas Wong, and we have implemented it as a backup plan. Though achieving satisfied results, it could still be improved in terms of resolving dissonances. It is true that music rules can be modularize by mathematics, however, running a chord analysis program on computer is somehow hard to achieve due to the fact that it is hard to find a way to perfectly generalize a rule for all possible music scores. Thus our methods could still be improved. We sincerely hope that our Hidden Markov Model and A.S.H.E.S. could be developed in the future, and we do have plans and directions on how to do that, which are elaborated in Section 8.

8. Future Work

8.1 Adjustment of HMM Model

As mentioned in section 4.8 *Possible Improvement in the Future*, more studies are needed for a proper way of choosing the time unit for the hidden markov model. Apart from choosing time unit, our choice of 12-dimensional vector as input format does not reflects the duration of each notes, or illustrating the neighborhood relation of adjacent nodes. The scheme for converting notes into vector could be modified if that factor is going to be concerned.

8.2 Proposing Harmonic Analyzer

Another two challenges that could raise at the same time when we are trying to modify our HMM is to identify harmony and to resolve dissonances. If an efficient harmonic analyzer could be applied and help our model define the right time unit, the accuracy of our model would be likely to get improved. The section ***HarmonicAnalyzer*** from previous year's group KY1601 could be considered as a referral.

8.3 Collecting More Training Data for HMM

As mentioned in Section 4.8.2 ,due to the limitation of time and accessible resources, we did not find much labeled classic music scores to train our model. If more data could be collected, it is likely that, our Hidden Markov Model could have a better performance, or it would be easier for us to pinpoint the problem and fine tune the model.

9. Work Distribution

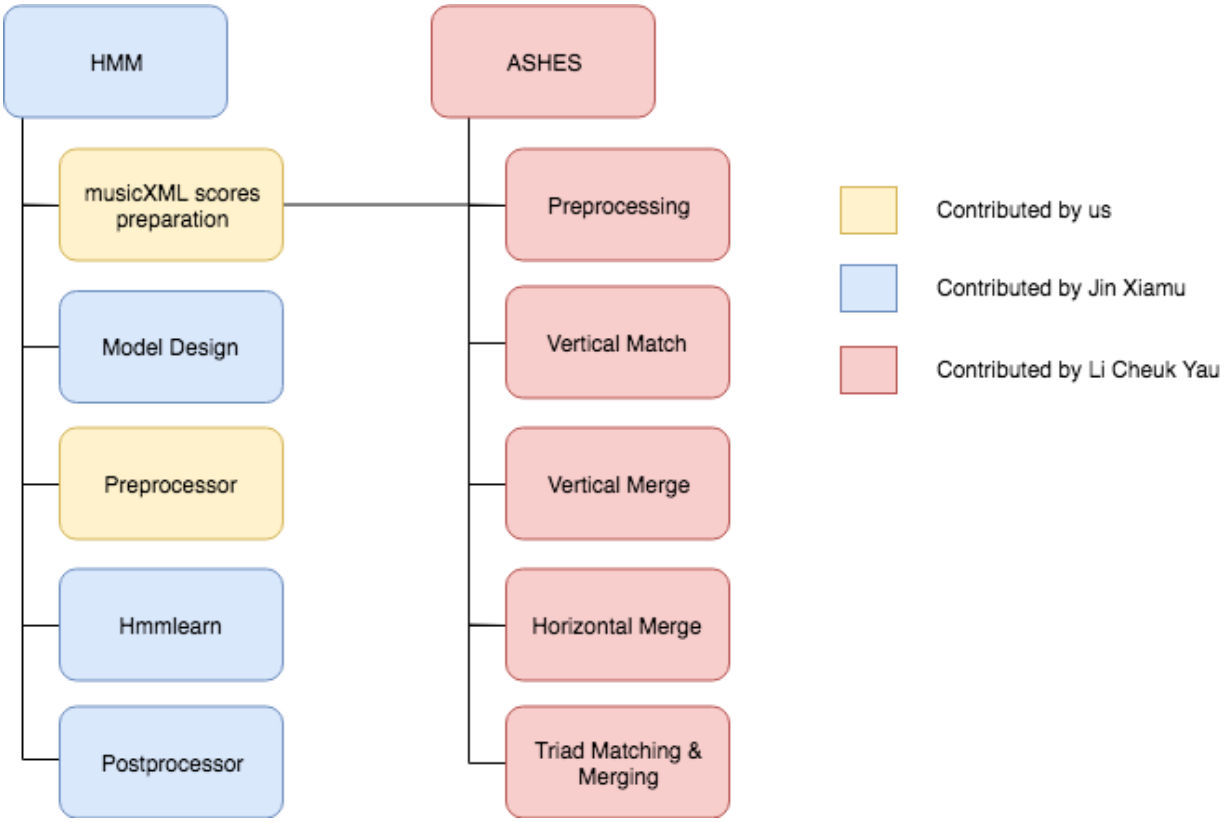


Figure 9. Distribution of Work

In the whole system, I collaborate with my teammate, Li Cheuk Yau, and we have distributed works as the above diagram shows. I was mainly in charge of designing and testing HMM model, while my teammate was mainly responsible in implementing A.S.H.E.S in this semester. I designed the HMM model myself, and after discussion with Professor Kevin Yip and Professor Lucas Wong during serval progress report meetings. I have finalized the model design and put it into practice by using the hmmlearn toolkit. During the process, Li Cheuk Yau helped me prepared input musicXML scores and write the **Preprocessor** program. For the rest of the parts of HMM, including training, testing and result analysis are mainly finished by myself.

10. References

The following open-source libraries are used in this project:
music21 - http://web.mit.edu/music21/

hmmlearn - https://hmmlearn.readthedocs.io

The following academic papers are referenced in this project report:
Chen, R., Shen, W., Srinivasamurthy, A., & Chordia, P. (2012, October). Chord Recognition Using Duration-explicit Hidden Markov Models. In ISMIR (pp. 445-450).

Berget, G. E. (2017). Using Hidden Markov Models for Musical Chord Prediction (Master's thesis, NTNU).

11. Appendix

11.1 Common Chords Handled in the Project

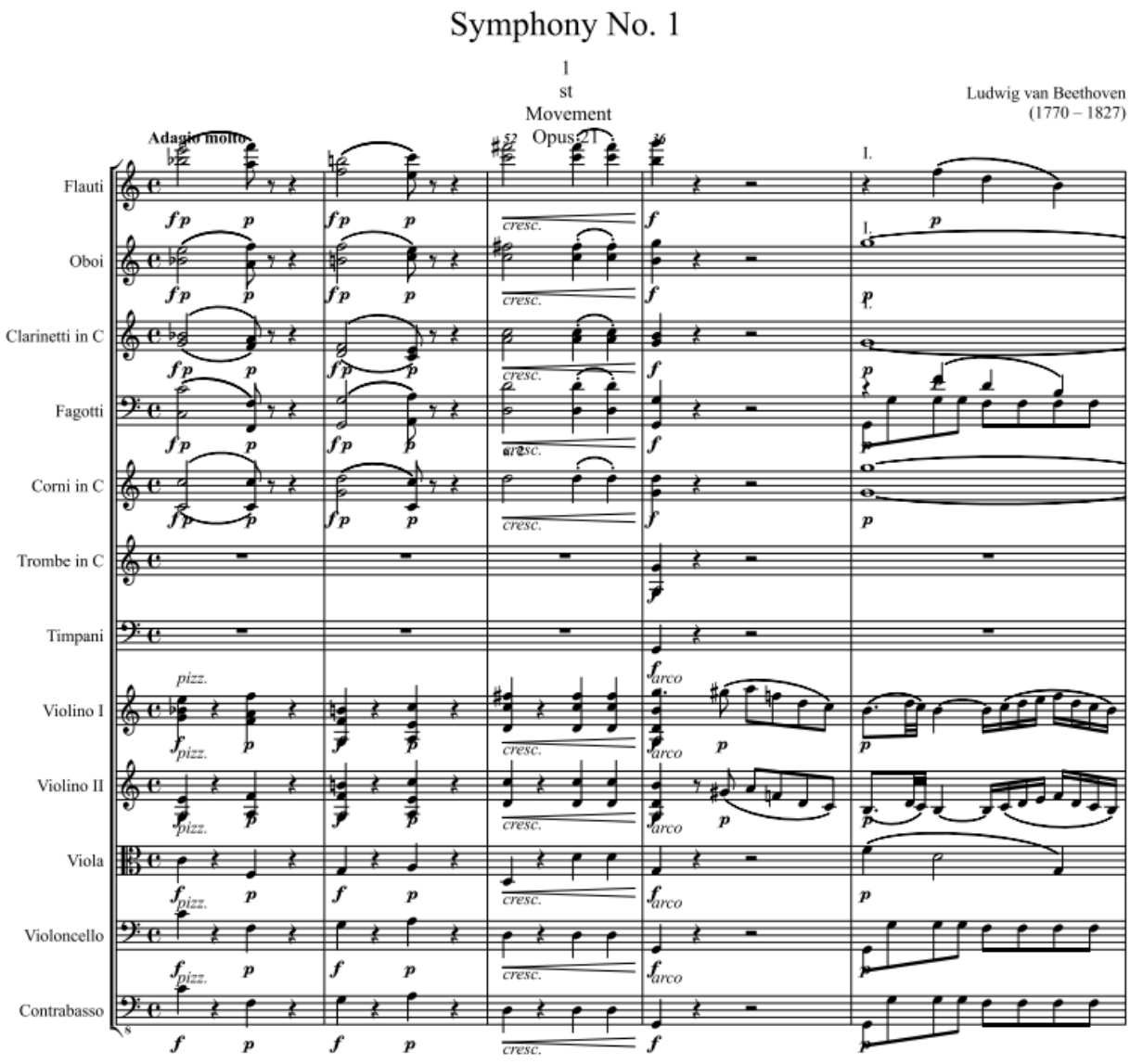
Common Major Chord Handled

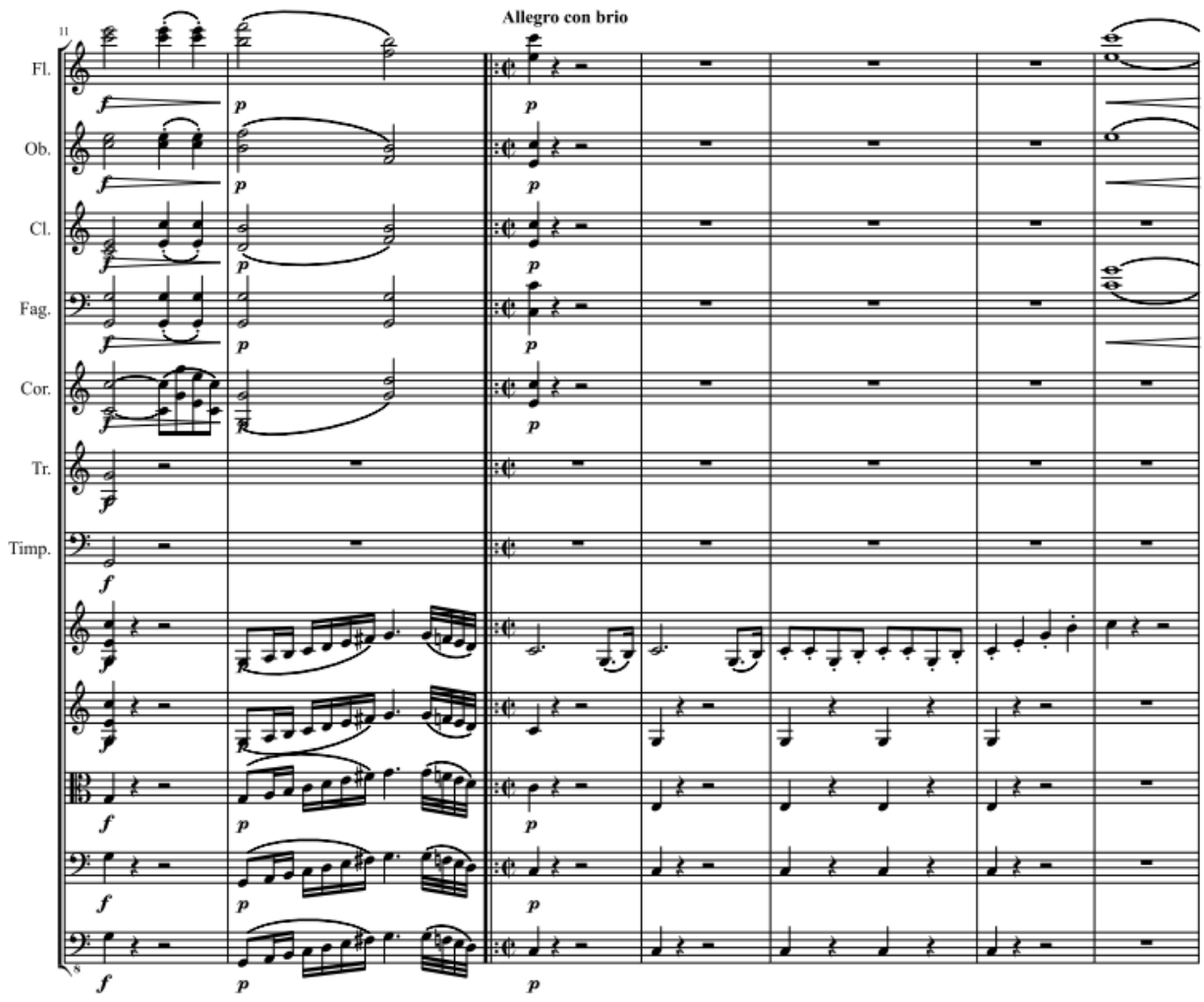
Chord Index Number	Chord Name	Roman	Possible Inversions	Root Note	2nd Note	3rd Note	4th Note	Chord Function	Chord Group
0	I	I	53, 63, 64	P1	M3	P5		Tonic	0
1	I7	I	7, 65, 43, 42	P1	M3	P5	M7	Tonic	0
2	♭II	bII	53, 63, 64	m2	P4	m6		Undefined	1
3	II	II	53, 63, 64	M2	P4	M6		Subdominant	2
4	II7	II	7, 65, 43, 42	M2	P4	M6	P1	Subdominant	2
5	III	III	53, 63, 64	M3	P5	M7		Tonic	3
6	III7	III	7, 65, 43, 42	M3	P5	M7	M2	Undefined	3
7	IV	IV	53, 63, 64	P4	M6	P1		Subdominant	4
8	IV7	IV	7, 65, 43, 42	P4	M6	P1	M3	Undefined	4
9	V	V	53, 63, 64	P5	M7	M2		Dominant	5
10	V7	V	7, 65, 43, 42	P5	M7	M2	P4	Dominant	5
11	♭VI	bVI	53, 63, 64	m6	P1	m3		Subdominant	6
12	German VI	bVI	65, 64, 43, 42	m6	P1	m3	A4	Subdominant	6
13	French VI	bVI	643, 642, 43, 42	m6	P1	M2	A4	Subdominant	6
14	Italian VI	bVI	63, 64, 53	m6	P1	A4		Subdominant	6
15	VI	VI	53, 63, 64	M6	P1	M3		Tonic	7
16	VI7	VI	7, 65, 43, 42	M6	P1	M3	P5	Tonic	7
17	VII	Vii	53, 63, 64	M7	M2	P4		Dominant	8
18	VII7	VII	7, 65, 43, 42	M7	M2	P4	M6	Dominant	8
19	Diminish VII7	VII	7, 65, 43, 42	M7	M2	P4	m6	Dominant	8

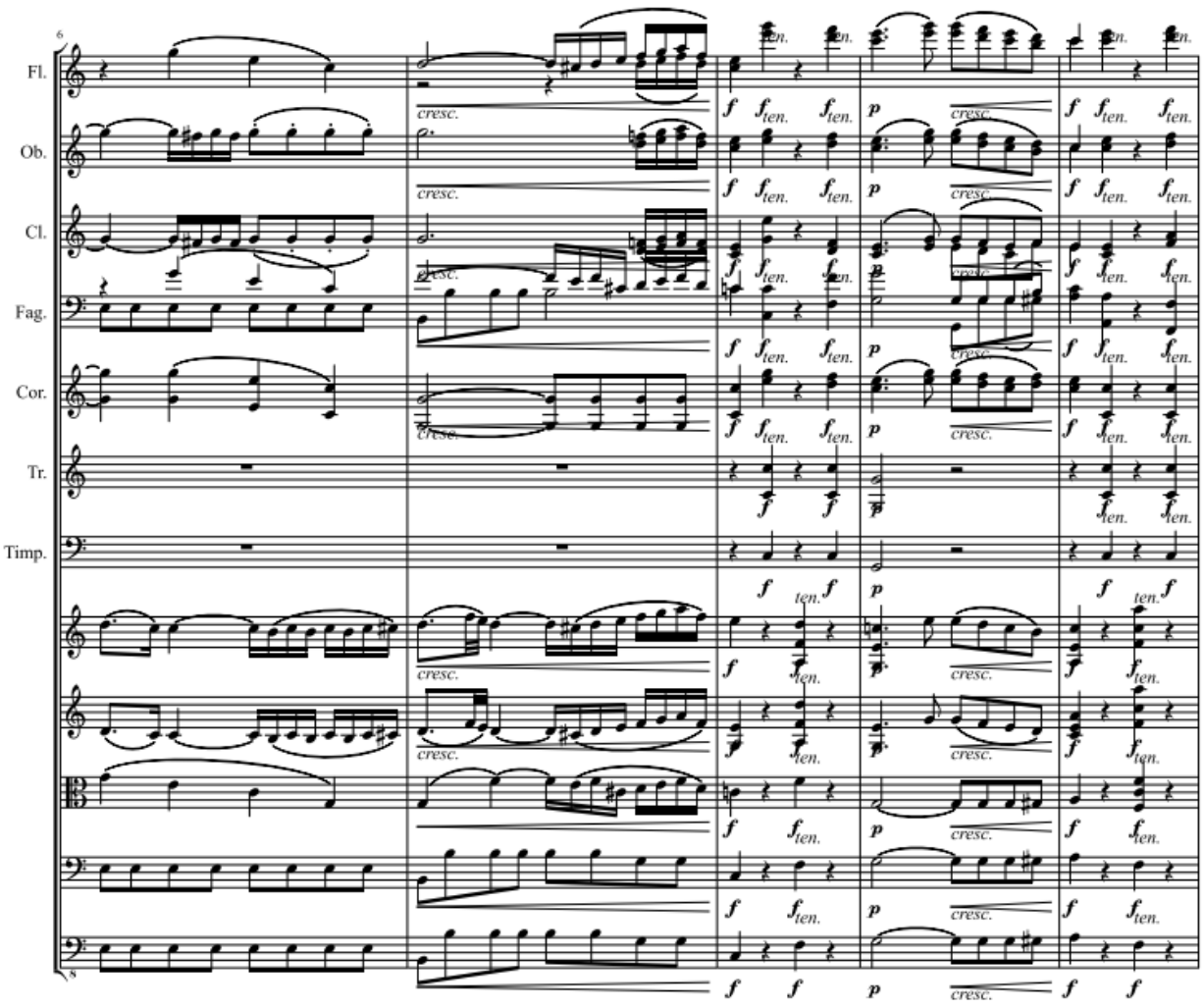
Common Minor Chord Handled

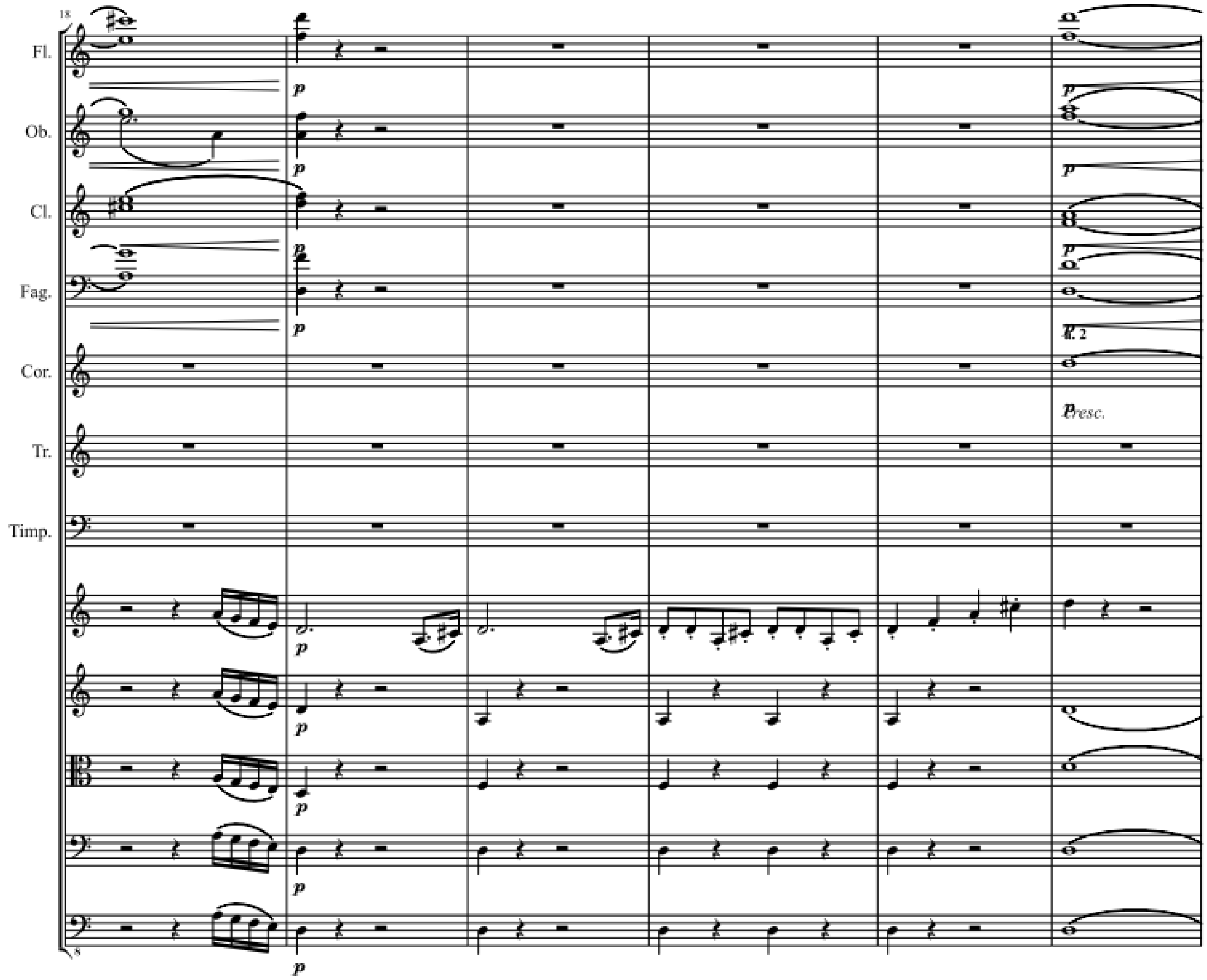
Chord Index Number	Chord Name	Roman	Possible Inversions	Root Note	2nd Note	3rd Note	4th Note	Chord Function	Chord Group
0	I	I	53, 63, 64	P1	m3	P5		Tonic	0
1	I+	I	53, 63, 64	P1	M3	P5		Tonic	0
2	♭II	bII	53, 63, 64	m2	P4	m6		Undefined	1
3	II	II	53, 63, 64	M2	P4	m6		Subdominant	2
4	II7	II	7, 65, 43, 42	M2	P4	m6	P1	Subdominant	2
5	III	bIII	53, 63, 64	m3	P5	m7		Tonic	3
6	IV	IV	53, 63, 64	P4	m6	P1		Subdominant	4
7	IV+	IV	7, 65, 43, 42	P4	M6	P1		Subdominant	4
8	V	V	53, 63, 64	P5	m7	M2		Dominant	5
9	V+	V	53, 63, 64	P5	M7	M2		Dominant	5
10	V+7	V	7, 65, 43, 42	P5	M7	M2	P4	Dominant	5
11	VI	bVI	53, 63, 64	m6	P1	m3		Subdominant	6
12	German VI	bVI	65, 64, 43, 42	m6	P1	m3	A4	Subdominant	6
13	French VI	bVI	643, 642, 43, 42	m6	P1	M2	A4	Subdominant	6
14	Italian VI	bVI	63, 64, 53	m6	P1	A4		Subdominant	6
15	VII	bVII	53, 63, 64	m7	M2	P4		Dominant	7
16	Diminish VII	VII	53, 63, 64	M7	M2	P4		Dominant	8
17	Diminish VII7	VII	7, 65, 43, 42	M7	M2	P4	m6	Dominant	8

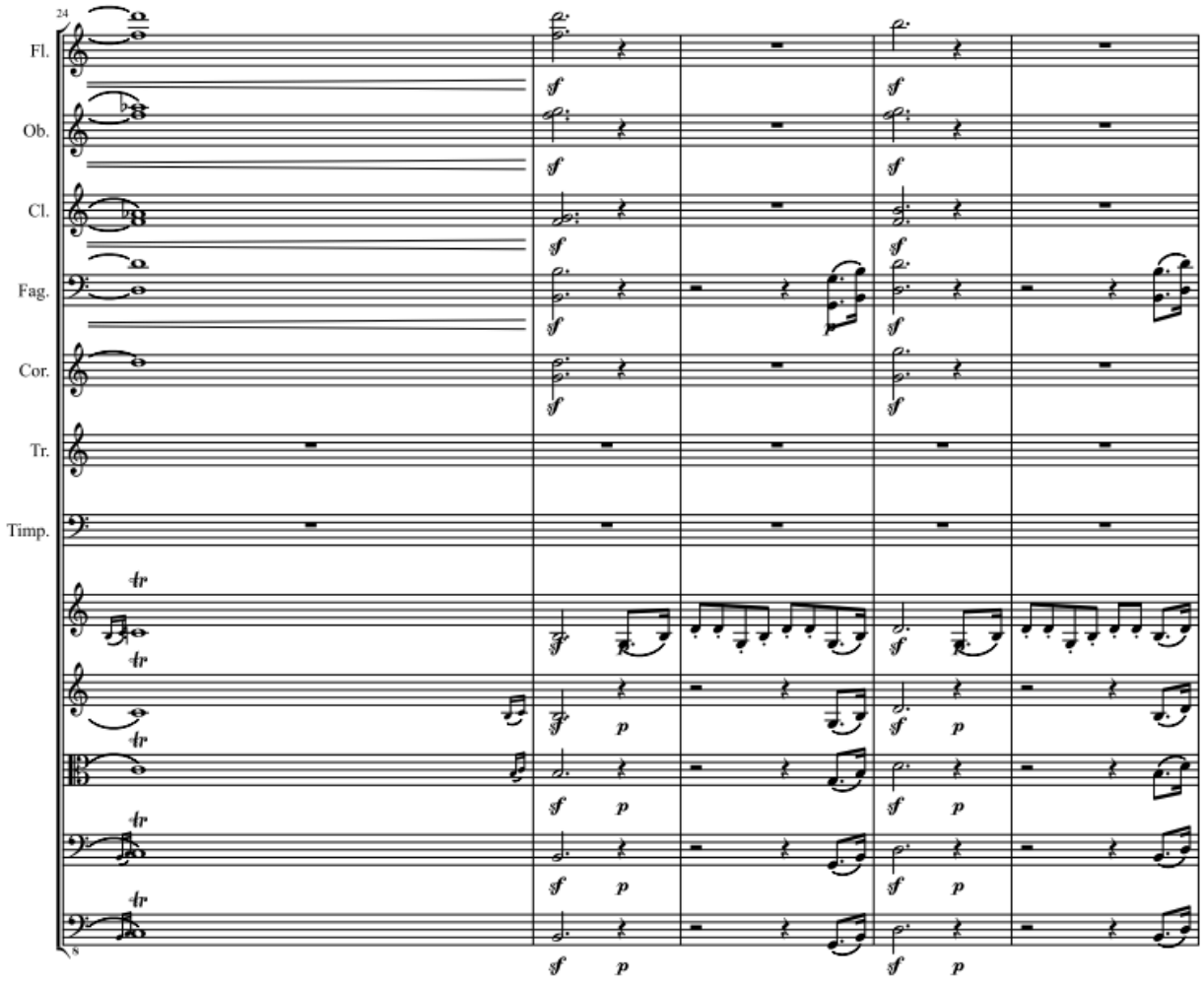
11.2 Beethoven Symphony No. 1

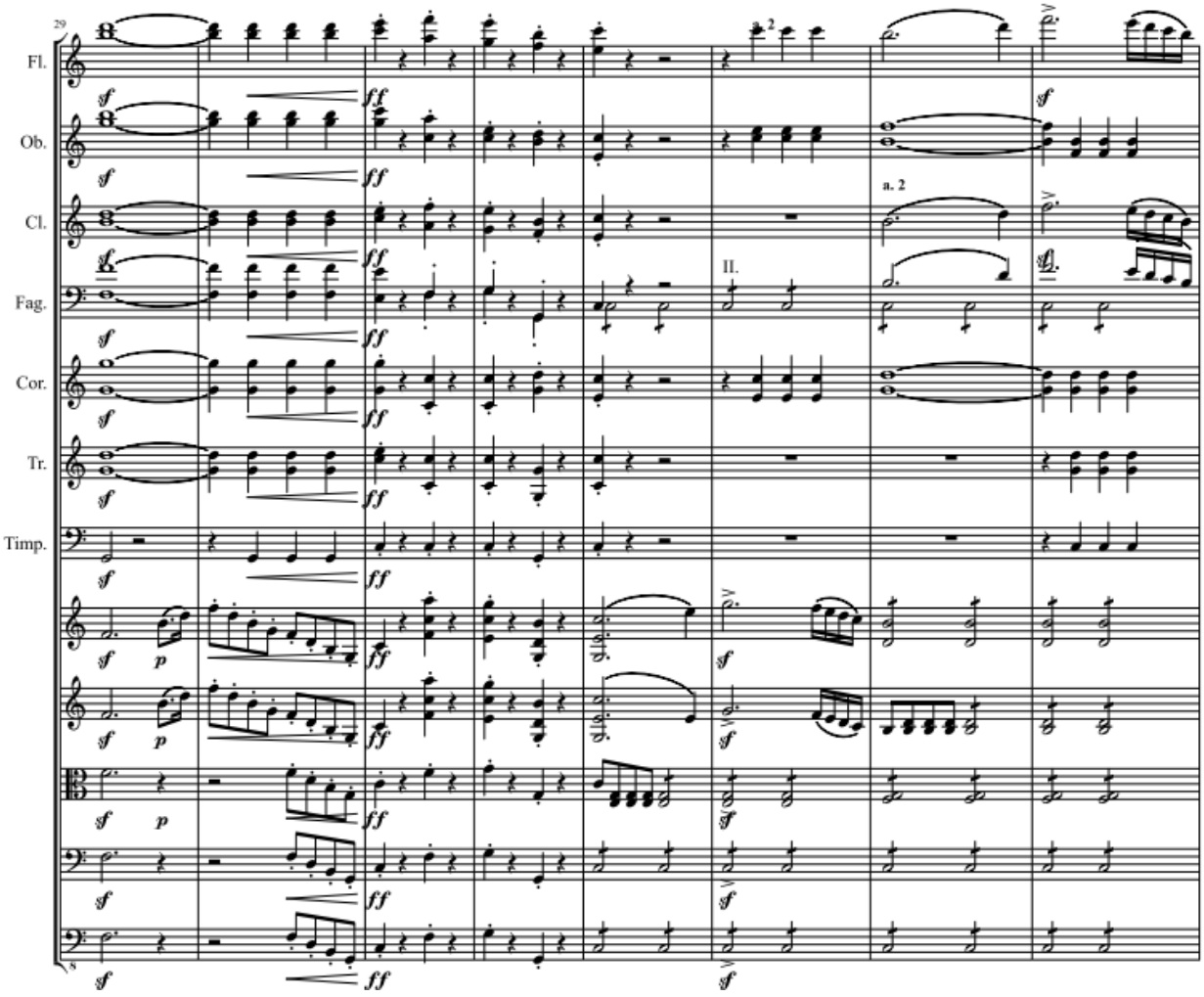



Allegro con brio
11
Fl.
Ob.
Cl.
Fag.
Cor.
Tr.
Timp.

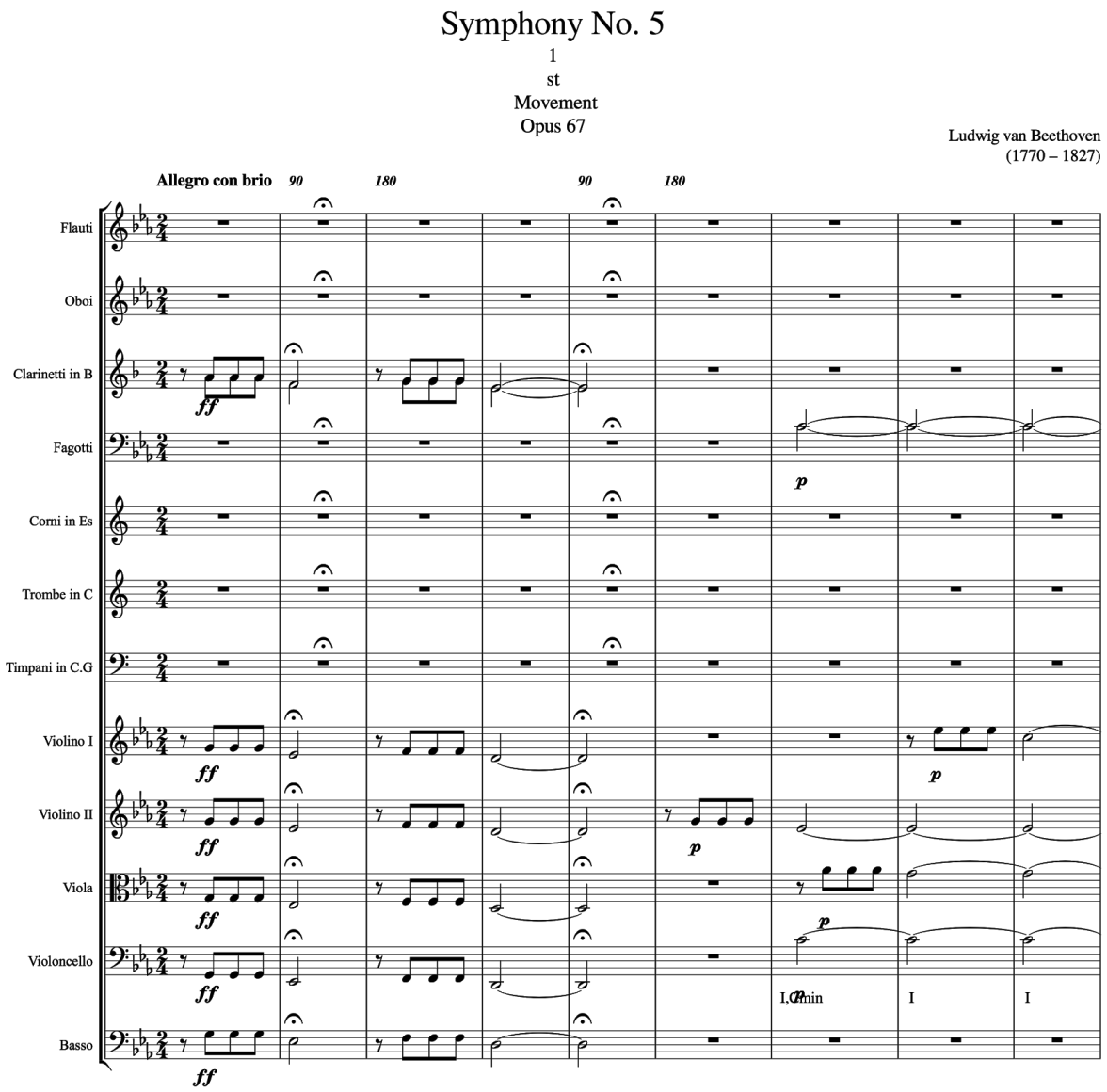

6
Fl.
Ob.
Cl.
Fag.
Cor.
Tr.
Timp.
cresc.
f
ten.
p

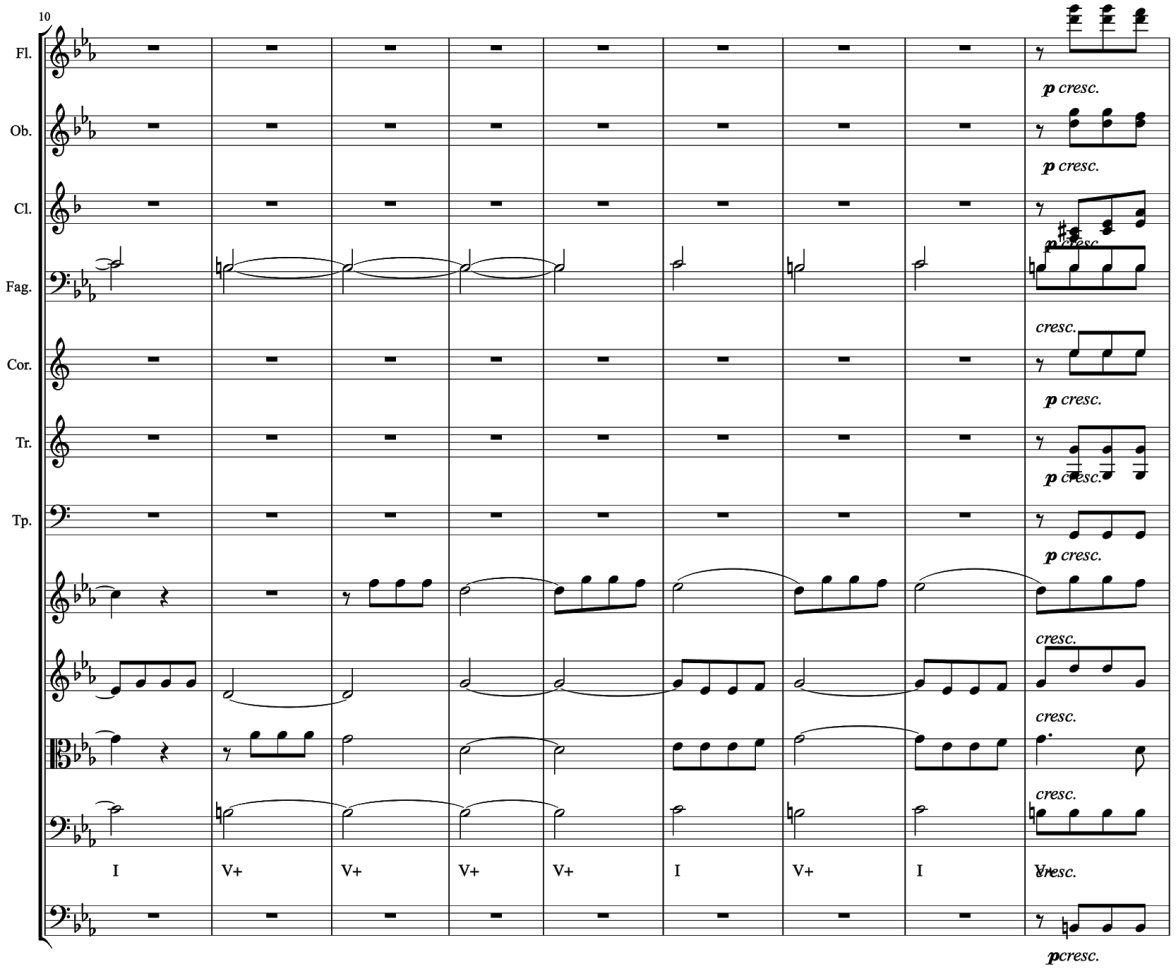

18
Fl.
Ob.
Cl.
Fag.
Cor.
Tr.
Timp.
p
cresc.

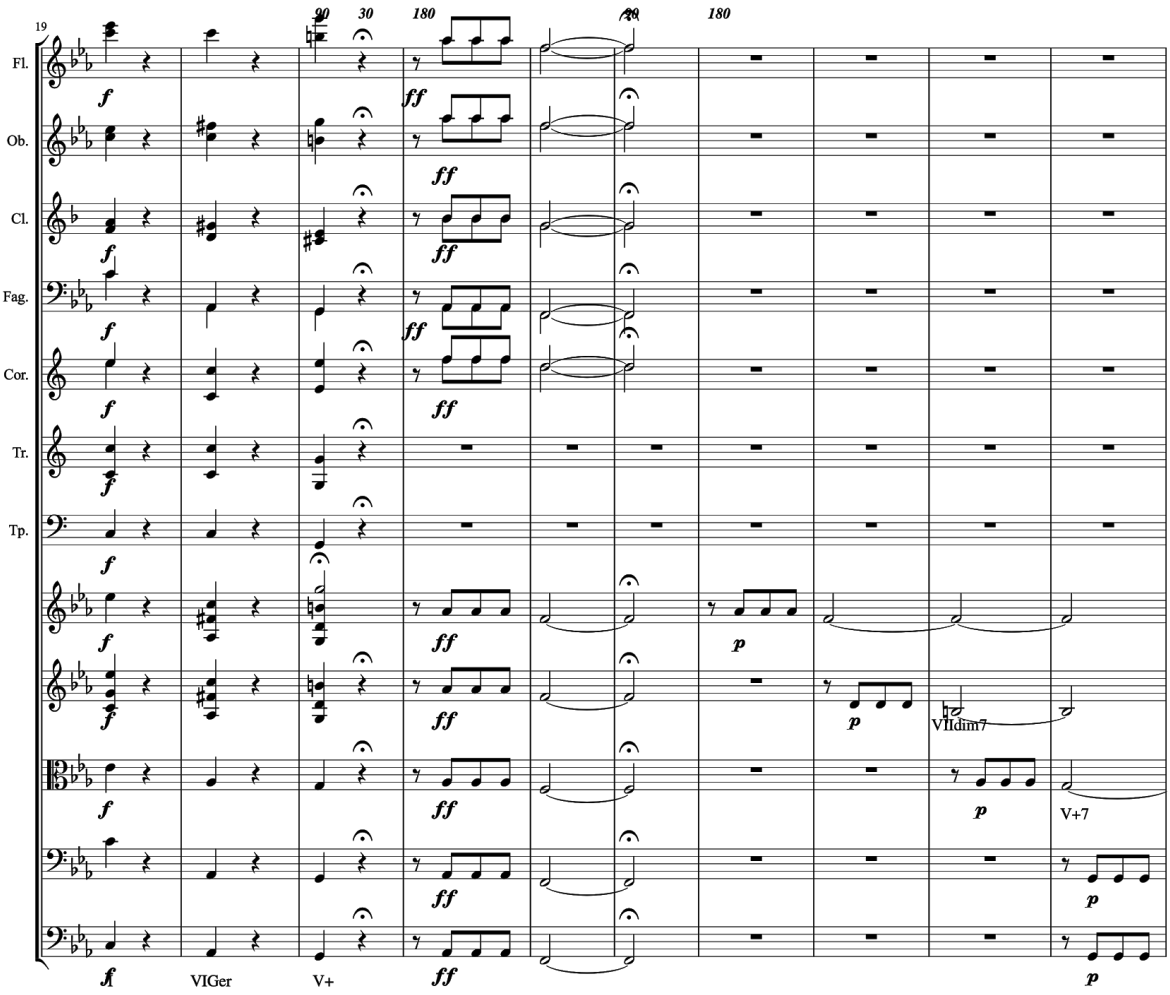

24
Fl.
Ob.
Cl.
Fag.
Cor.
Tr.
Timp.
8

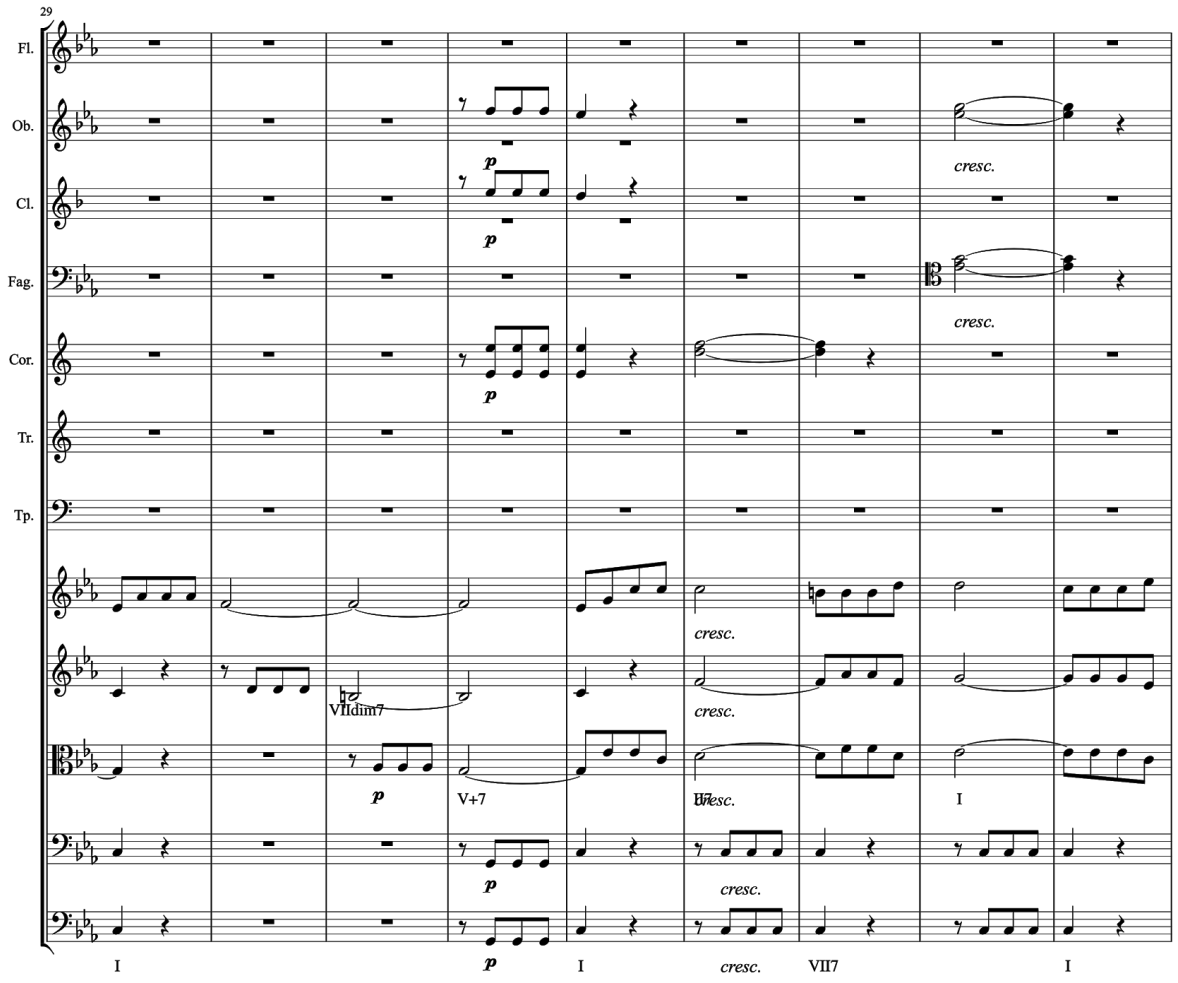

29
Fl.
Ob.
Cl.
Fag.
Cor.
Tr.
Timp.
a. 2
II.

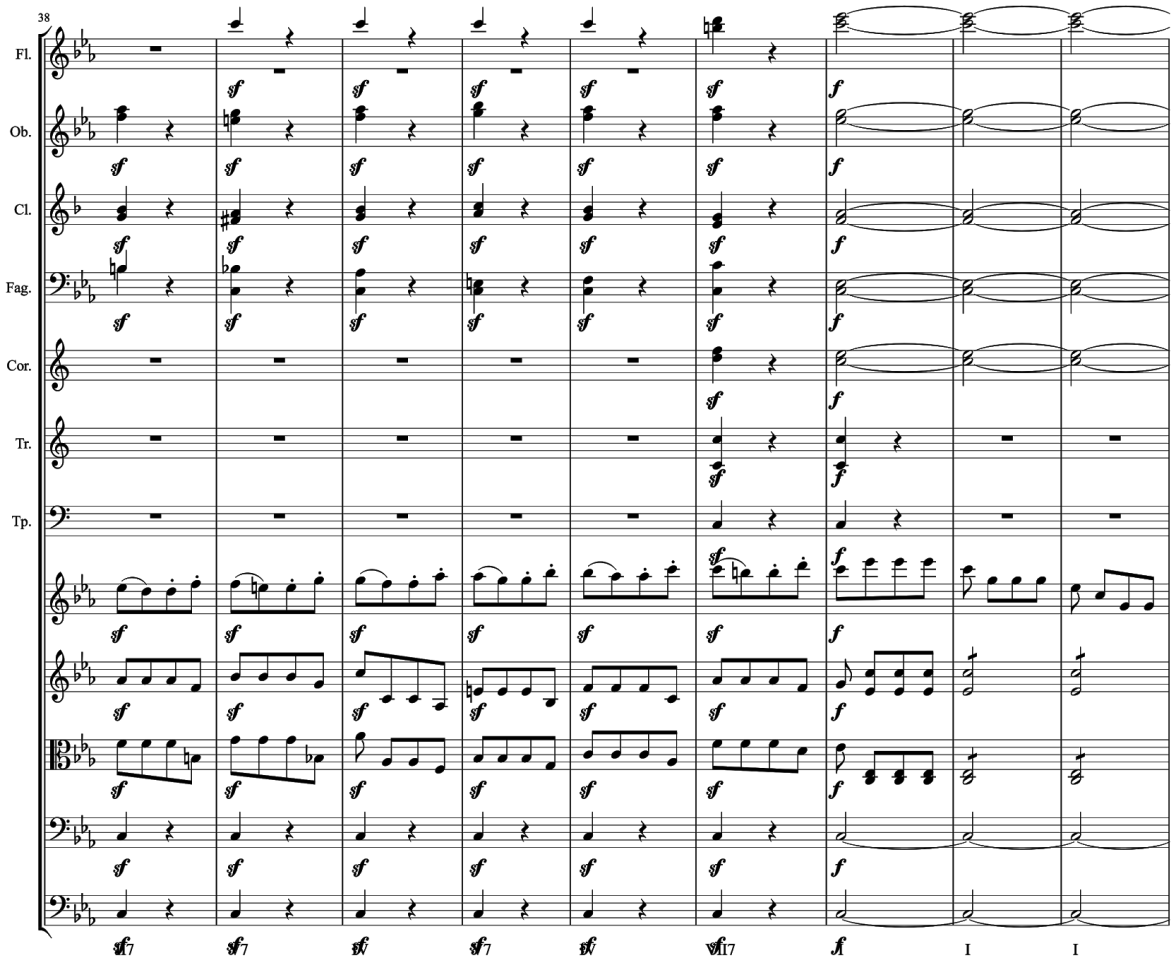
11.3 Beethoven Symphony No. 1

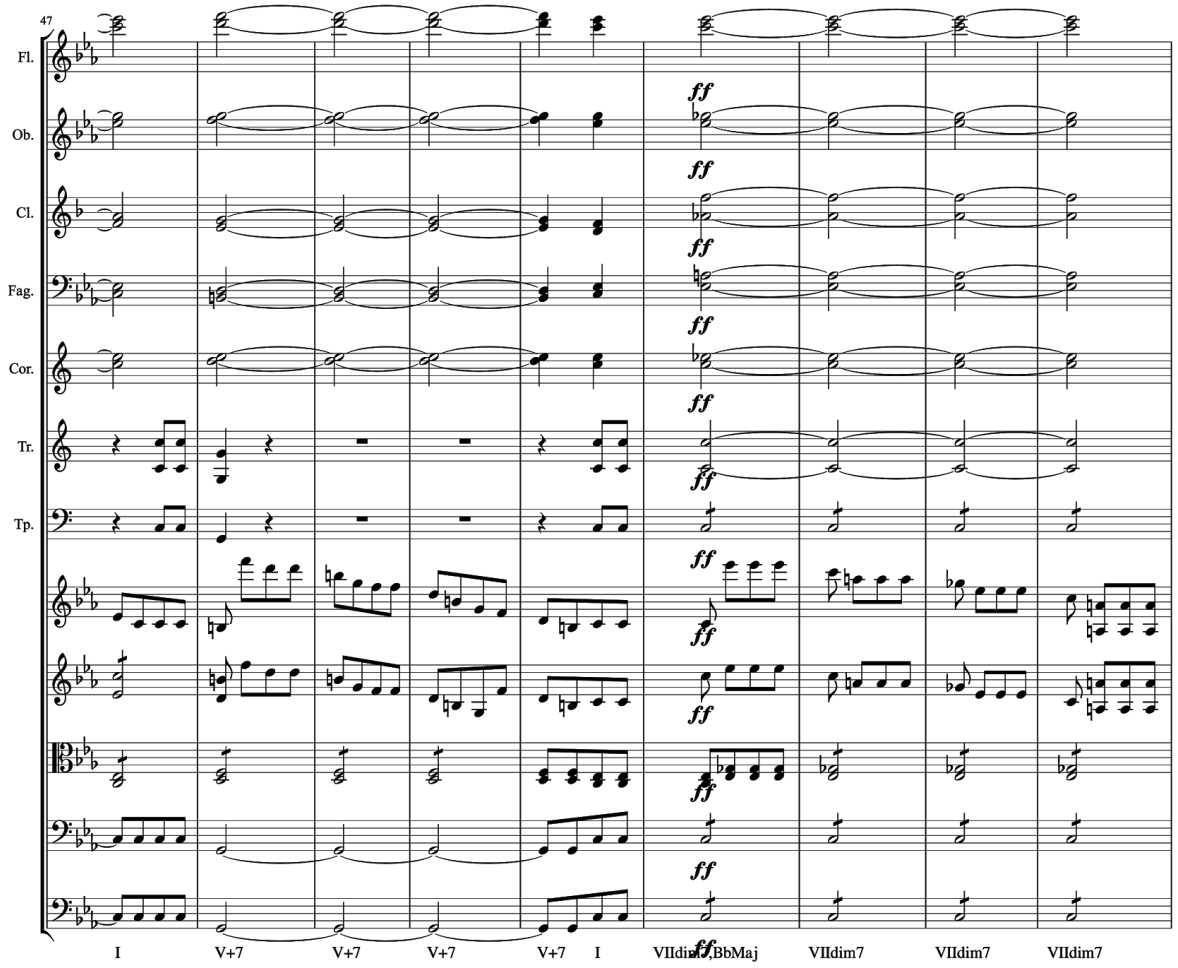



10
Fl.
Ob.
Cl.
Fag.
Cor.
Tr.
Tp.
p cresc.
cresc.
I
V+
V+
V+
V+
I
V+
I
V+


19
90
30
180
20
180
Fl.
Ob.
Cl.
Fag.
Cor.
Tr.
Tp.
VIIdim7
V+7
VIGer
V+


29
Fl.
Ob.
Cl.
Fag.
Cor.
Tr.
Tp.
cresc.
VIIdim7
V+7
I
I
cresc.
VII7
I


38
Fl.
Ob.
Cl.
Fag.
Cor.
Tr.
Tp.
I
I


47
Fl.
Ob.
Cl.
Fag.
Cor.
Tr.
Tp.
ff
I
V+7
V+7
V+7
V+7
I
VIIdim7/BbMaj
VIIdim7
VIIdim7
VIIdim7