Creating a Tool for Facilitating and Researching Human Annotation of Musical Patterns

Master Thesis

Stephan Wells

Supervised by

Anja Volk

Judith Masthoff

Iris Yuping Ren

Universiteit Utrecht

A thesis presented for the degree of Game & Media Technology

Graduate School of Natural Sciences
Utrecht University

Utrecht, 2019

Abstract

The use of musical patterns is highly widespread in all varieties of music, and we define them as noteworthy segments that repeat in a given piece of music. Annotations of such patterns are valuable in many problems tackled by the field of music information retrieval, such as music similarity, classification, prediction, and labelling, and these annotations can be procured from domain experts or algorithmically. Unfortunately, there is a distinct lack of domain expert annotations, in part due to the laborious process of annotating by hand and consequently digitising these annotations for use in music information retrieval systems. Any automatic musical pattern annotation method that is data-driven needs a large number of annotations to be trained on, but there is an absence of digital infrastructure in place for manually annotating music to produce these annotations.

In this project, we introduce a novel software designed for musical pattern annotation. The software allows users to load a music file and easily and intuitively annotate repeating segments in the piece of music, with the option to save their annotations to a file for use in future research applications. We perform a user study where we prompt users to annotate patterns in classical music excerpts and take a survey on their experience with the tool, analyse the resulting annotations and survey data, and perform a comprehensive evaluation of the tool and the annotations created with it.

This work was supported by the Tertiary Education Scholarship Scheme (Malta, 2019).

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Preface

We live in an age of information, where data is constantly generated and disseminated and the demand to analyse, process, and structure it into a form that can be understood by users is ever increasing. Unfortunately, regardless of the domain, data is seldom useful in its raw form. For instance, it is trivial to write a computer vision algorithm that is able to state how many pixels of a certain colour there are in an image, but to develop an algorithm that is able to describe semantically what objects are present within the image is a much more difficult task. In essence, we want our information retrieval systems to develop a more complex model of the data that is being processed, such that this model can be utilised to retrieve more meaningful information (objects in an image file, melodies in an audio file, etc) than simply subsets or statistics of the raw data.

This is where the real bottleneck of information retrieval lies: pattern recognition. We wish to automate what humans do so effectively, which is the ability to detect and learn patterns in the world around us (Jain, Duin, & Mao, 2000). Humans identify patterns everywhere, from extremely basic domains such as a stream of binary symbols (Feldman, 1961), to complex object recognition in the visual domain (DiCarlo, Zoccolan, & Rust, 2012), and music is no exception (Deutsch & Feroe, 1981; Krumhansl & Jusczyk, 1990). As such, many approaches to automate pattern recognition in music have been developed with varying levels of success (Ren, Koops, Volk, & Swierstra, 2017), but this endeavour faces many challenges, such as a notable absence of existing musical pattern data to work with (Volk & Van Kranenburg, 2012). Thus, while pattern recognition has made great strides in domains such as text, images, video, and speech, its application in music is still very much an open research problem.

The goal of this thesis is to explore and facilitate human annotation of patterns in musical pieces. Due to there being a lack of digital infrastructure for manual annotation of musical patterns, we introduce and describe the development of a novel software tool designed to allow annotators to intuitively discover, annotate, and share their annotations of musical patterns. To evaluate this piece of software, a comprehensive user study

is carried out where the tool is used by a number of participants to produce pattern annotations on musical excerpts. The tool is evaluated qualitatively through a survey and annotations created with it are evaluated through various statistical measures. Finally, we show that manual annotation using our software possesses many benefits over the traditional pen-and-paper method of musical annotation.

The structure of this chapter is as follows: the next section will give some background on the field being tackled by this thesis. We will then discuss musical pattern annotation in more detail, outline its challenges, and compare different approaches of pattern annotation. Following this, the problem definition and research contribution is presented.

1.2 Music Information Retrieval

Before discussing music information retrieval (often shorted to MIR), it is pertinent to describe the difference between data and information. In (Zins, 2007), leading scholars in areas related to information science discuss the difference between the two terms. A common understanding among the experts involved in this research is that data can be defined as raw, unorganised, and still needing to be processed, whereas information is the result of contextually meaningful processing of data. For example, the heights of every school student can be considered data; heights are unstructured, measured figures. Conversely, the average of all of the heights of every male student would be considered information; the raw data is processed to extract meaningful conclusions.

Information retrieval deals with this very process, i.e. retrieving information from raw data. It is what drives search engines, recommendation systems, statistical analyses, and other powerful applications of data processing. In the context of music, we can note many examples of musical data being processed to glean music information, such as using pitch or chroma data to arrive to melodic information in melody transcription (Poliner et al., 2007) or using amplitude data to detect onsets/transients and extract harmonic changepoints for musical note information (Hainsworth & Macleod, 2003). Music information retrieval, therefore, is concerned with the process of transitioning from low-level music data (frequency, amplitude, timbre, symbolic notes, music metadata, etc) to high-level music information (melody, harmony, rhythm, structure, genre, etc).

In this thesis, we are concerned with a particular form of music data, which is symbolic music data[1], and the goal is to facilitate the gathering of melodic information in the form of musical patterns. In the next section, we will discuss the concept of a musical pattern.

1.3 Patterns in Music

The primary focus of the research performed by this thesis is the discovery and annotation of patterns in music, and this section will provide a comprehensive background on the

[1]The difference between symbolic and other forms of music data is discussed in **Section 2.2**.

topic, its prevalence in the literature, and the challenges present in the field.

1.3.1 Definition

What is a musical pattern? In previous texts, one can come across various definitions that attempt to answer this question. In (Hsu, Liu, & Chen, 2001, p. 1), a pattern is defined as "a sequence of notes which appears more than once in a music object". The MIREX task that deals with discovery of repeated segments (Collins, 2019) echoes this definition, defining a pattern in much the same way as "a set of ontime-pitch pairs that occurs at least twice in a piece of music". In both of these definitions, we can note that repetition is a key element of a musical pattern. If a sequence of notes does not appear more than once in a piece of music, then it is implied that it is not one worth considering as a salient piece of data, in the context of music pattern recognition. Thus, we define a pattern as a noteworthy or characteristic melodic segment of a piece of music that is identified to have multiple exact or inexact repetitions throughout the piece. An annotation, on the other hand, is simply a marked observation of an occurrence of a pattern within a piece of music (see **Figure 1.1**).

A contemporary, real-world example of repetition in music is the use of leitmotif (Bribitzer-Stull, 2015), which is a central and recurrent theme in a musical composition that evokes or describes a key idea, emotion, or situation that the piece is attempting to convey. Characteristic, repeating melodic patterns such as the leitmotif can be seen as part of the fundamental structure of a piece of music (Bent & Drabkin, 1987) and thus can be used for various applications in music information retrieval, such as modelling music similarity and classification. This is the primary motivation for studying them and attempting to automate their discovery and annotation.

Of course, patterns need not be just segments of melodies. As described in a section on musical patterns and their representations in (Herbert & Sumner, 1993), patterns in music are often multi-dimensional, and one can discover patterns in a music's dynamics, rhythm, orchestration, harmony, or many other dimensions. However, for the purposes of this thesis, we are concerned with patterns of the melodic kind.

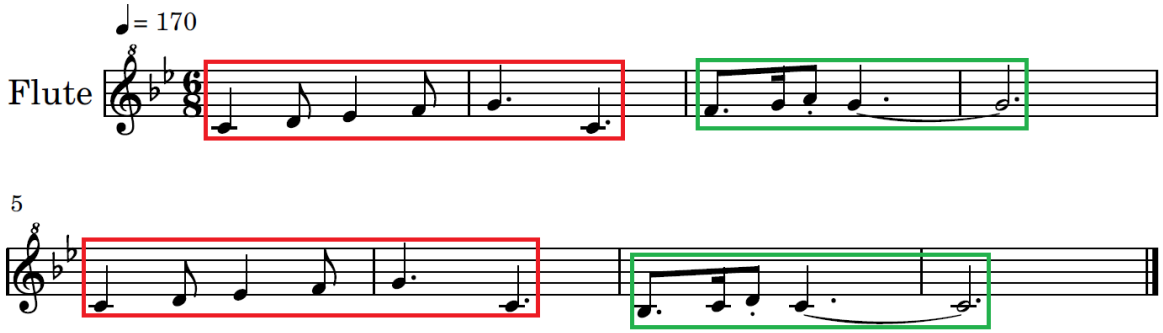


Figure 1.1: An example of two different patterns (red and green), each with two occurrences, being annotated on original sheet music.

Figure 1.1 shows a very basic musical excerpt[2] with short melodic pattern occurrences being annotated using red and green bounding boxes. This is a simple example showcasing exact repetition (red bounding boxes) and inexact repetition (green bounding boxes). The occurrences marked in red repeat exactly because they contain the same notes, whereas the occurrences marked in green repeat inexactly because the second occurrence is transposed downwards from the first occurrence.

1.3.2 Challenges

Pattern recognition in music is challenging for many reasons, but one of the foremost obstacles is the subjectivity that is inherent in the process of discovering patterns. Much like many other branches of music information retrieval where human processing is involved, pattern recognition and annotation is liable to disagreements among annotators.

This subjectivity has been shown in various instances in previous research, such as in (Collins, Laney, Willis, & Garthwaite, 2011), a study on pattern importance in musical excerpts, where music students rated already-discovered patterns, with fluctuating levels of agreement, on how important or characteristic they are in the context of the music. Furthermore, in (Meredith, 2015), a paper on analysis of pattern discovery approaches, an exceptionally stringent argument on the ambiguity inherent in exiting human pattern annotation data is presented. There are many controversies that human annotators must face when attempting to annotate patterns in music, such as: When does a pattern start and end? How long should a pattern be at minimum? Have I found all occurrences of a given pattern in this piece? How similar must a repetition of a pattern be for me to count it?

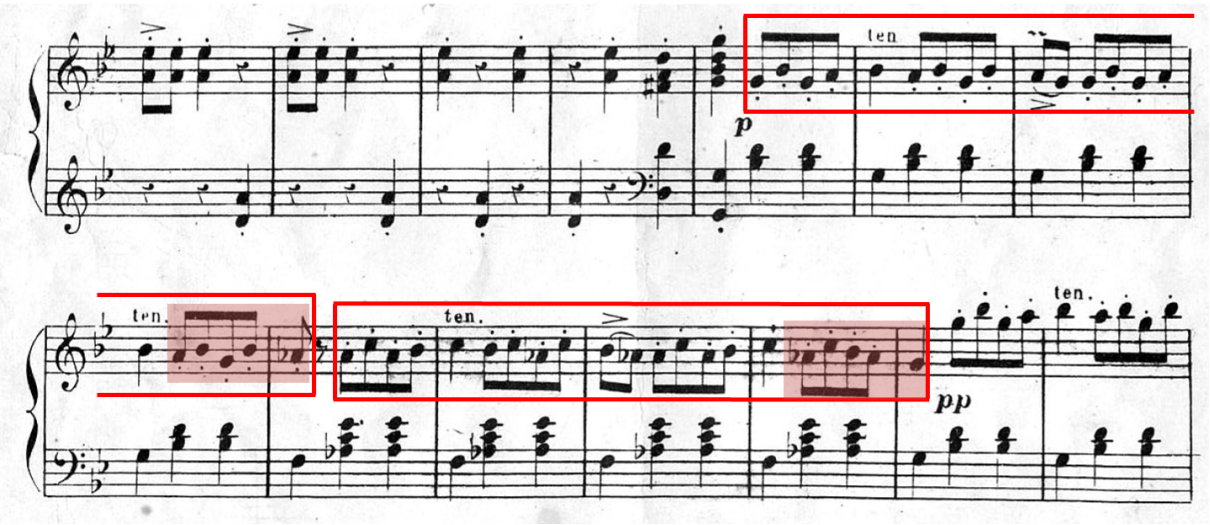


Figure 1.2: An excerpt from Saint-Saëns' Danse Macabre with two occurrences of a pattern annotated using red bounding boxes and a notable melodic difference in each occurrence shaded red.

[2]Descriptions of musical terms and representations will be touched on in **Section 2.1**.

To give an example, consider **Figure 1.2**, which is an excerpt of *Danse Macabre* (Saint-Saëns, 1872). Two pattern occurrences have been annotated using bounding boxes. The second occurrence is highly similar to the first with merely transposed notes, but there is a small difference towards the end of the pattern (shaded red). This is an example of inexact repetition that causes ambiguity and can be annotated differently by different annotators. For instance, some annotators may choose to omit the shaded segment from their annotations due to the inexactness of the repetition. Others may include it as it marks the end of a phrase and the similarity is high enough for it to be considered a part of the pattern. In short, there are multiple arguably valid ways of annotating these segments.

This presents us with a problem: If humans are unable to reach a consensus when discovering and annotating musical patterns, then how can we generate any kind of usable *ground truth* data to use for data-driven, automatic approaches to pattern discovery? This is certainly a difficult problem, but it is not an insoluble one. In (Reidsma & Op Den Akker, 2008), a paper on subjective annotations, an experiment is performed which, while unrelated to music, deals with and studies subjective annotations done by humans on videos depicting group conversations from the 100-hour AMI meeting corpus (Carletta et al., 2005). Annotations were done on cues that were visible from these videos, such as glances, speech intonation, and so on, and were classified by the authors of the paper into different kinds of classes, namely *manifest content* (directly observable events), *pattern latent content* (events that need to be inferred from observations), and *projective latent content* (events that require subjective interpretation). They note that most of the disagreements come from pattern/projective latent content (such as discerning whether or not a line of speech is directed at a specific person or at the entire group) as opposed to manifest content, and they introduce the concept of *consensus objectivity*, which is the idea that, while annotations are inherently subjective, finding the overlap between different annotator's interpretations where they agree is a viable method for dealing with subjectivity. The idea that inter-annotator agreement is something to strive for is backed up in (Meredith, 2015), where the author challenges the notion that we should attach significance to human-annotated patterns that are not endorsed by a large number of domain experts.

Ultimately, human annotations in a subjective domain such as music will always be subject at least partially to interpretation, but the research stresses the notion that subjective annotations are not necessarily unusable, as long as precise annotation schema are used and the annotations themselves are found to be agreed upon by a large number of annotators. The next section will compare and contrast human-annotated patterns with automatically-discovered patterns, reviewing the research in this area.

1.3.3 Human Annotation vs Automatic Discovery

Automatic pattern discovery, which is the process of automating the discovery and marking of patterns in a given piece of music, is a significant focus point in the literature

(Meredith, 2015; Lattner, Grachten, & Widmer, 2018; Ren et al., 2017; Hsu et al., 2001; Janssen, De Haas, Volk, & Van Kranenburg, 2013). That being said, annotations done by humans are still studied extensively (Volk & Van Kranenburg, 2012; Collins et al., 2011; Ren, Volk, Swierstra, & Veltkamp, 2018; Ren, Nieto, Koops, Volk, & Swierstra, 2018), and it would be insightful to understand what the differences between the two annotation approaches and their results are.

Of course, automatic pattern annotation is naturally more scalable and feasible for real-world application. It is difficult to justify having humans laboriously annotate repeated segments of music when a computer algorithm would be able to perform this ostensibly trivial task in a fraction of the time. Unfortunately, pattern annotation is indeed not trivial, and there is much room for improvement in how we train automatic solutions to identify patterns. In fact, it has been noted that many existing pattern discovery algorithms simply find too many patterns in a given piece of music (Meredith, 2015; Boot, Volk, & De Haas, 2016; Ren et al., 2017), whereas human experts would find only a few patterns.

There is a concept of *salience* in pattern discovery, where some repeated segments in music are more perceptually significant than others. Early research in the field notes that it is certainly possible to design an automatic method to find all possible exactly-repeating segments in a sequence of notes, but the vast majority of those are not salient (Meredith, Lemström, & Wiggins, 2002). Unfortunately, the salience of a pattern is challenging to formally characterise, and the lack of an objective ground truth in annotations of this manner makes it difficult for one to train an automatic method to determine the so-called salience of a discovered pattern.

Having said that, the work of (Boot et al., 2016) is concerned with exactly this issue. In their paper, they study the relevance of repeated patterns in music for specific retrieval tasks, namely modelling similarity[3] and compression[4]. Using a symbolic data set of Dutch folk song music, they compare the patterns annotated by musicological experts to patterns extracted automatically using multiple state-of-the-art automatic pattern discovery methods. Upon obtaining these pattern annotations, their approach to compare them was to use them for music compression through sparse melody construction[5] and then training classifiers that attempt to classify these sparse melodies into their respective *tune families* (Bayard, 1950). They found that repeated occurrences of musical patterns were relevant for the similarity and compression retrieval tasks but that the human annotations performed significantly better in both compression ratio and classification accuracy.

It is clear that, at least in the tasks of compression and similarity on folk music, automatic pattern annotations cannot compete with human-generated annotations. There is a veritable discrepancy in the two kinds of annotations, and this is supported by the research

[3]Modelling music similarity involves attempting to create a model that can quantify the measure of resemblance between different pieces of music.

[4]Music compression in this context is the act of reducing the size of a symbolic music file by representing it in a more condensed format.

[5]Sparse melody construction is the process of removing notes from a musical piece that do not belong to any discovered pattern, thus compressing the music.

done in (Ren, Volk, et al., 2018), where they successfully train classifiers to distinguish between human-found pattern annotations, automatic pattern annotations, and random passages. They show that, indeed, we are not yet at a stage where we can automate the discovery of patterns that are indistinguishable to those found by humans, despite automatic pattern annotations being at least more meaningful than random passages.

In general, humans appear to be better at making sensible decisions about the salience of patterns in music. Humans annotate far fewer patterns than automatic methods, and the patterns annotated by humans appear to be more analytically interesting or descriptive/characteristic of a musical piece than patterns discovered by an automatic method. Therefore, at the current state that the research is in, gathering and studying human annotations is still a fruitful and necessary venture.

1.4 Problem Definition

We have defined the concept of a musical pattern, outlined the challenges, and provided motivation for the importance of human pattern annotations. This brings us to the problem that this thesis is concerned with: there is a distinct lack of digital infrastructure to enable and support human annotation of music. Currently, patterns are annotated by humans primarily through the traditional *pen-and-paper* approach, where sheet music is printed out and patterns are annotated using handwriting, drawing, and underlining.

Figure 1.3 below shows an example of patterns being annotated on sheet music using a traditional pen-and-paper method, taken from the HEMAN data set (Ren, Nieto, et al., 2018). In this photo, patterns are identified, circled, and marked with a number next to each circle which indicates the relevance of the motif to the entire excerpt.

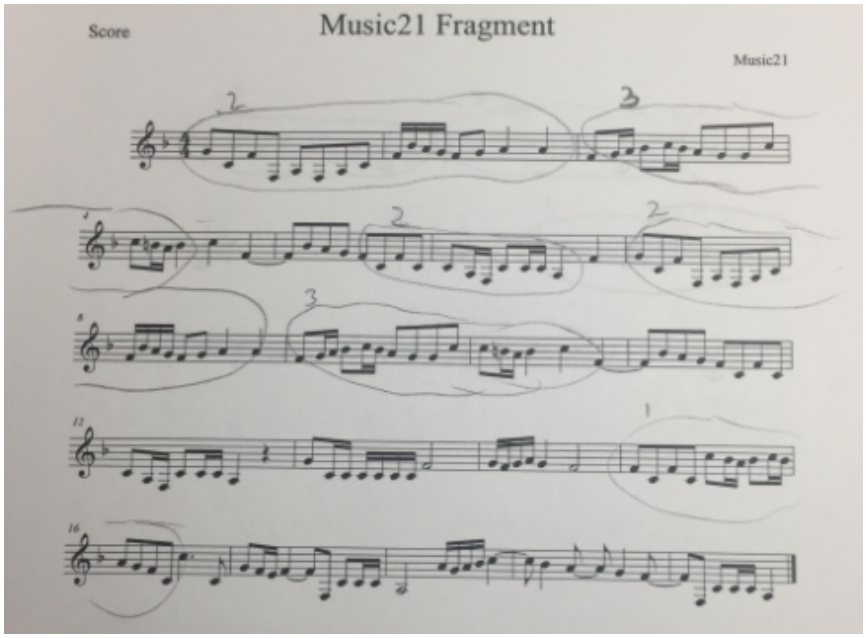


Figure 1.3: An example of pen-and-paper annotation on patterns in a musical excerpt.

Needless to say, there are clear drawbacks with this approach:

- Most significantly, a large amount of overhead is required. Papers need to be printed, prepared, and distributed. They need to then be collected, and any annotations done in an incorrect or illegible format must be discarded. Finally, for them to be useful in a research setting, they likely need to be digitised in some form, which is a menial and cumbersome process.
- It is impossible to listen to the sheet music unless it is played on an instrument or separate digital music files are provided corresponding to the sheet music pieces.
- There is a lack of customisability and functionality; basic quality-of-life features that might be present in a digital annotation software program are not present in this traditional method.
- The method of selecting the exact notes to mark as a pattern is imprecise and subject to misinterpretation. For instance, consider bar 16 of **Figure 1.3**; the notes of the pattern are circled roughly using a pencil, but the circle goes through the final note halfway. It is uncertain whether or not the annotator intended to include this note in the pattern.

The above disadvantages of the pen-and-paper method make it very apparent that it is not scalable, difficult to distribute to people, liable to errors, and overly time-consuming. In fact, in (Ren, Nieto, et al., 2018), a paper on musical pattern ambiguity, it is explicitly stated that the pen-and-paper approach they employed proved time-consuming and that a digital solution would be a great boon for human pattern annotation. Unfortunately, there is a shortage of publicly available software that is suited to the task of pattern annotation, which is the primary motivation that this thesis aims to address.

1.5 Contribution

The research contribution of this thesis project is the development, evaluation, and research surrounding the creation of a functional annotation software tool: ANOMIC. ANOMIC is a software tool aimed at music annotators that deals with identifying, viewing, and saving musical pattern annotations. The tool operates through a piano roll interface[6], runs on Windows machines, and boasts an interactive annotation process with various customisability options and quality-of-life features such as showing/hiding patterns, finding exact occurrences of a pattern, loading/saving annotations, and audio playback.

This thesis describes the implementation and features of ANOMIC in detail and performs a comprehensive evaluation of the tool by means of a user study. Through this research, we show that the tool is positively received among various demographics, highlighting its strengths and shortcomings, and we analyse a large data set of musical pattern annotations done using ANOMIC, demonstrating how inter-annotator agreement

[6]More details on different interfaces of symbolic music can be found in **Section 2.2**.

is improved using this tool and its features compared to more traditional annotation approaches.

The upcoming chapter will provide the necessary background knowledge to fully grasp the topics of the thesis, namely a background in basic musical terms, technical knowledge, and similar work in this field. The next chapter deals with the methodology and will go in detail on design choices, ANOMIC's interface, implementation specifics, technical specifications, algorithm explanations, user study details, and so on. The following chapter lists and discusses the results of the user study, performing an evaluation on the annotations generated by the participants. The final chapter then discusses and recapitulates conclusions, future work, and possible improvements.

Chapter 2

Background

The purpose of this chapter is to familiarise the reader with the necessary terminology and background knowledge to adequately understand the content of the upcoming chapters.

The domain of this research is, of course, music, so **Section 2.1** will provide any readers unfamiliar with music theory some basic terminology and concepts to attain a working musical knowledge for adequate comprehension of this thesis. **Section 2.2** will give an overview of the more technical details involved in this thesis, such as different representations of audio, different symbolic music interfaces, and so on. Finally, **Section 2.3** will go over related work in previous literature that attempts to provide digital solutions to human pattern annotation.

2.1 Musical Background

This section provides knowledge on basic music terminology and a surface-level understanding of sheet music to allow a fuller understanding of the more nuanced music-related details of this thesis.

2.1.1 Terminology

Various musical jargon will be mentioned in this thesis. Refer to **Appendix C** for a complete list of working definitions of the musical terminology used in this paper. This section will briefly explain the key terms and concepts.

The most important music-related concept to understand is the musical note. A note is a representation of a sound that plays at a particular pitch for a certain duration. Notes can be played by any pitched instruments and can be depicted using a wide variety of representations, though this thesis will focus on two such representations: traditional sheet music and digital MIDI-based representation. When multiple notes are placed together in sequence, these form a melody. Melodies can be any length and can have pauses (rests) between the notes. Conversely, when multiple notes play at the same time, this forms a

chord. A piece of music typically consists of multiple melodies alongside an underlying harmony that is made up of a progression of chords.

Another pertinent term to understand for this thesis is the concept of voices. A voice is a constituent, continuous part of music that can be perceived as independent from other notes being sounded at the same time. One would typically delegate voices to be played by different instruments or, say, different hands of a pianist. When multiple voices are present in a piece, we define this piece to contain *polyphony*. There are various different types of polyphony (monody, homophony, etc) which are distinguished by the role that each different voice performs in the piece, but for the scope of our research, we simply define polyphony to be the presence of multiple voices.

Finally, throughout this thesis, the term *transposition* is used often to refer to chromatic transposition of musical notes. Chromatic transposition is the process of considering a set of musical notes and altering the pitch of each one by an equal amount. Pitch differences in music are defined by intervals which, in Western music, are quantified as semitones, where a semitone is the minimum pitch disparity between two notes. For example, a repetition of a melody consisting of eight notes can be transposed upwards by an interval of seven semitones, meaning that each of the eight notes in the transposed melody will sound seven semitones higher than its corresponding note in the original melody.

2.1.2 Sheet Music

Sheet music is the written form of music that utilises symbols and notation to symbolise musical elements such as notes, dynamics, and performance directions. Printed sheet music dates back as early as the late 15th century (Hyatt King, 1964) and is a well-established format that is taught to musicians to this day. With the advent of computer software, there are additional methods of representing music, but sheet music is the conventional method for printing, performances, and teaching.

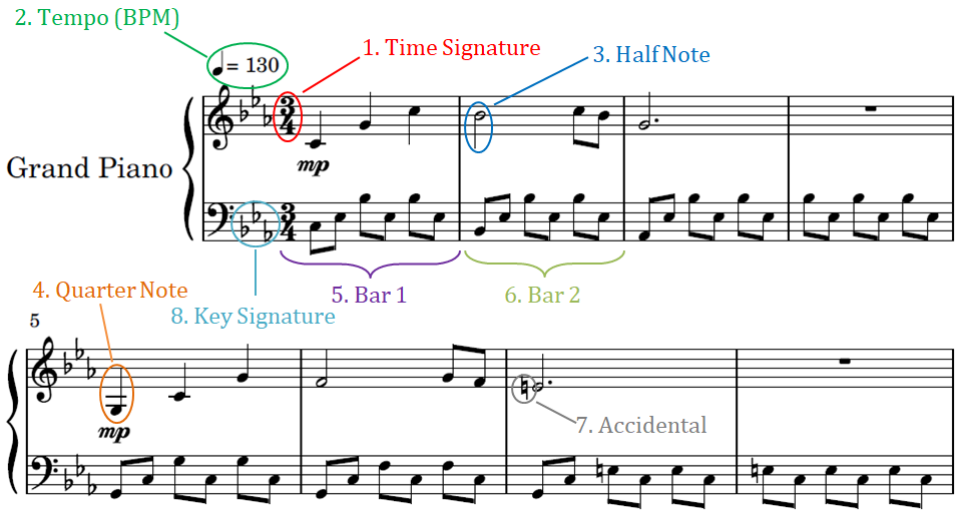


Figure 2.1: A labelled example of sheet music.

Figure 2.1 provides an example of original sheet music with many of the constituent elements described in the previous section labelled. In this example, the time signature (1) is 3/4, which essentially means that there are three beats in each bar and each beat has the duration of a quarter note. The tempo (2) is 130 BPM[1], implying that the music should be played or performed in such a way that 130 quarter-note beats would fit in a minute. There are two examples of notes labelled in the figure: a half note (3) and a quarter note (4). The vertical position on the stave (the horizontal lines) indicates the pitch of the notes. Bar 1 (5) and bar 2 (6) are also labelled, and the excerpt in total has eight bars.

An accidental (7), or more specifically a natural sign, can be seen next to one of the notes (an E note). Despite the key signature (8) specifying that any note that is an E should be lowered in pitch by a semitone, making the note E flat, the natural sign cancels that alteration, making the E note an E natural. **Figure 2.2** is another labelled excerpt of the sheet music to showcase some additional elements that make up typical sheet music notation.

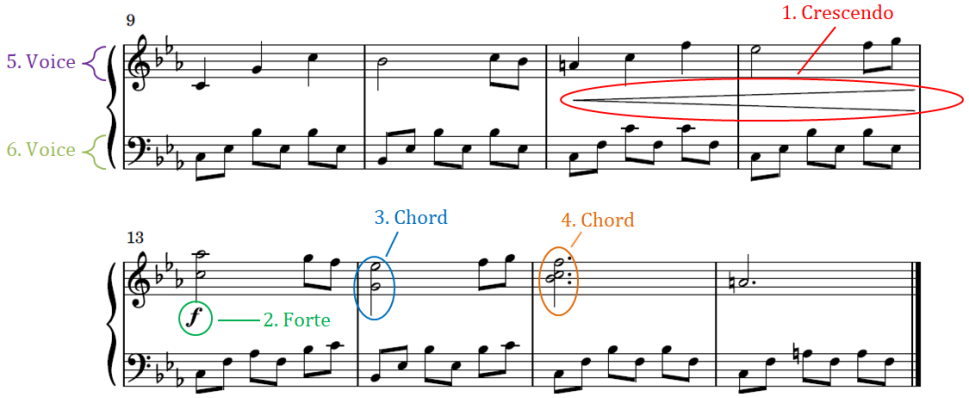


Figure 2.2: Another labelled example of sheet music.

In this figure, we see some examples of dynamics in the form of a *crescendo* (1), implying a gradual increase in loudness, and a *forte* symbol (2), implying that any music following that symbol is to be played or performed loudly. There are also examples of a two-note chord (3) and a three-note chord (4). Finally, the music that is notated in the figure is said to be polyphonic, because it has two voices, with the first voice (5) being played by the right hand of the pianist, carrying most of the music's melody, and and the second voice (6) being played by the left hand of the pianist, where the melody's accompaniment can be found.

Many piano pieces consist of two voices, one played by either hand, but some works, especially those of the Baroque era, incorporate more voices, requiring finer control and independence to be performed using just two hands. For instance, **Figure 2.3** is an excerpt of a three-part Baroque piece (Bach, 1720), where you can see from the shaded areas that it is possible to extrapolate three independent voices from the music.

[1]BPM stands for beats per minute.

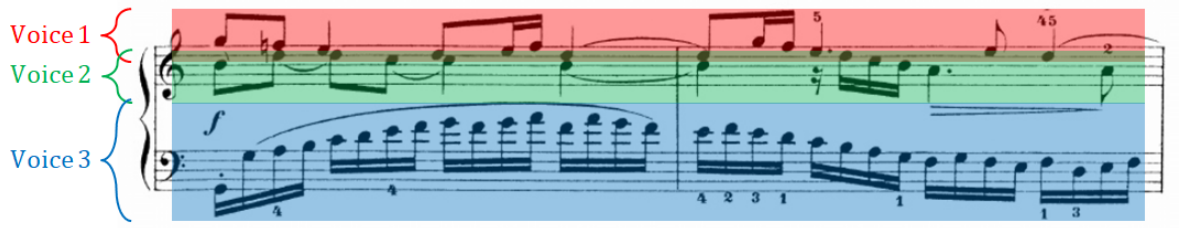


Figure 2.3: An excerpt from Bach's Sinfonia No. 1 in C Major BWV 787.

2.2 Technical Background

Now that the musical terminology and information has been considered, this next section will go into the more technical aspect of the thesis. We will start by explaining how music is stored digitally, outlining the difference between symbolic music and audio. Then, we will go into more detail on the MIDI symbolic music format and what kind of data it consists of. Afterwards, we will discuss different digital interfaces for viewing MIDI-formatted music. Finally, we will provide some relevant technical background pertaining to the JAMS file format (Humphrey et al., 2014), a JSON-based standard used for saving musical annotations.

2.2.1 Digital Music

In the past few decades, as digital media became more prevalent, it was a natural outcome that music would be among the many forms of media to be digitised. The many different formats of digital music can be classified into two distinct categories: audio files, or symbolic files. In this section, we will describe both types of music format and summarise their differences.

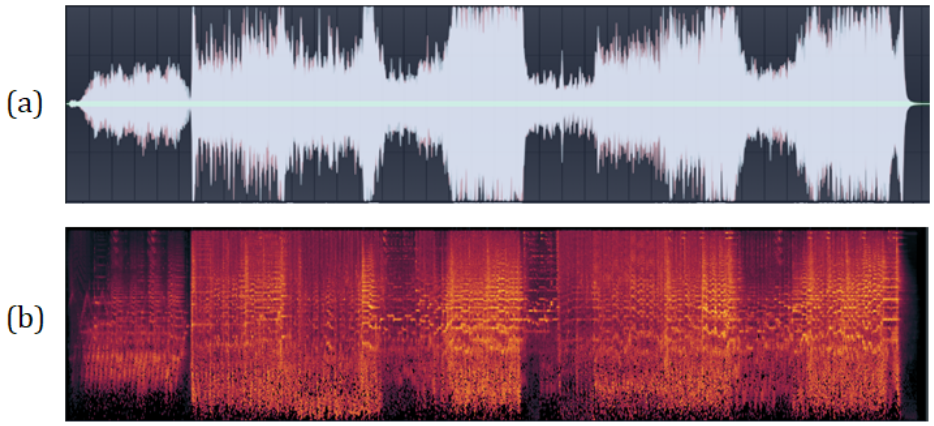


Figure 2.4: An example of digital audio in two representations: a waveform (a) and a spectrogram (b).

Digital audio files come in many varieties, with some of the most common raw audio file formats being .WAV (Waveform Audio File) and .AIFF (Audio Interchange File Format)[2]. These files are created using a technique referred to as pulse code modulation, which is a sampling technique that had applications in telephony and speech sampling (Black & Edson, 1947) before eventually being used for digital audio sampling. It operates by sampling the amplitude (loudness) of audio at specific time intervals (dictated by the *sampling rate*) and saving that sample as a binary number (whose number of bits is dictated by the *bit depth*). **Figure 2.4** shows two of the various representations of digital audio: a waveform and a spectrogram.

The above figure illustrates the data that can be found in a digital audio file. The waveform (a) is simply the amplitude of the sound wave sampled at a sample rate of 44.1 KHz with a bit depth of 16 bits. The spectrogram (b) is a frequency-time graph whose frequency information is calculated from performing a Fourier transform on the sampled audio data.

Digital audio files can sample any kind of audio, from music to speech to background ambience. While, through this method of digitisation, all of such audio can be sampled with excellent fidelity, a drawback of digital audio from the perspective of our research is that the data it contains is not readily usable in a musical context. It is, to this day, a challenging task to extract, for example, musical note information from an audio file of a song (a task known as music transcription (Gowrishankar & Bhajantri, 2016)), especially in polyphonic pieces or multi-instrument pieces. Evidently, to perform musical analyses, having the musical information at hand instead of having to extrapolate it from raw digital audio is vital.

This is where the symbolic music format shines. Instead of saving precisely-sampled data that remains faithful to the audio waveform of the music, symbolic music formats purely save data on the musical qualities of the music. This means that parsing through a symbolic music file will, among other data, give you musical notes, time signatures, tempos, and dynamics, as opposed to digital samples of audio. It is, of course, vastly limiting to store digital music by representing it as only musical information, as the sonic qualities and fidelity of digital audio is lost, but for performing musicological research, this allows for much deeper analyses into the actual musical qualities of a piece of music rather than the audio data of performances of that music.

The most widespread symbolic music format, and indeed the format that is relevant to this thesis, is the MIDI format. The next section describes this format in more detail.

2.2.2 The MIDI Format

The Musical Instrument Digital Interface (MIDI) standard (Moog, 1986) started as a means of unifying the protocol under which electronic music instruments (such as hard-

[2]Note that other digital formats exist, such as .MP3, .OGG, and .FLAC, but these often feature lossy or lossless compression of the audio through psychoacoustic techniques to reduce filesizes.

ware music synthesisers) would operate. Previously, every hardware brand would have their own formats, which, naturally, was considerably limiting the industry growth. As such, the MIDI format was devised and presented a way of representing the inputs of a hardware synthesiser as lightweight data that can be carried through specialised MIDI cables to be interpreted by another piece of hardware.

Most of the data that would have to be transferred over a cable would be note data, which generally consists of the pitch, start, end, and *velocity* of the note. The term "velocity" intuitively originates from the fact that hardware MIDI synthesisers and keyboards were often designed to be sensitive to the velocity with which the notes were being pressed by the performer. Higher velocities, or in other words, heavier force being exerted onto the MIDI hardware's buttons, corresponded to louder sounds, just like how pressing a traditional piano's keys harder would generate a louder sound. Thus, note velocities were coined as a method of working with musical dynamics.

This protocol translated very well to a symbolic file format, since it contained most of the information that a piece of music would need; pitches, durations, loudness, instrument channels, time signature, key signature, polyphony support, tempo, and many other forms of musical information. Of course, musical information alone does not generate music that a listener can hear; stating times and pitches of notes alone does not produce sound. This is why MIDI files need an associated *MIDI sound module* to interpret the MIDI data and actually play the notes. These sound modules would contain a basic set of synthesised musical instruments with which to play the musical notes. While sound modules were separate hardware devices in the past, nowadays software sound modules are used to interpret and play note data, with a low-quality sound module coming bundled by default with most modern audio drivers.

Describing the content of a MIDI file requires a basic understanding of computer code. At its most low-level form, data in a computer is in a binary format, i.e. it follows a base-2 numeral system, with a *bit* being the smallest possible unit of data that can be either `1` or `0`. Bits are interpreted in collections of eight bits called *bytes*, which are the smallest addressable unit of digital information in a computer memory. **Figure 2.5** shows the byte 01001101 with the weights above the binary digits and the MSB (1) and LSB (2) labelled. Simply put, decimal representation of the binary number can be attained by adding the weights of each `1` digit together, yielding a result of `77`.

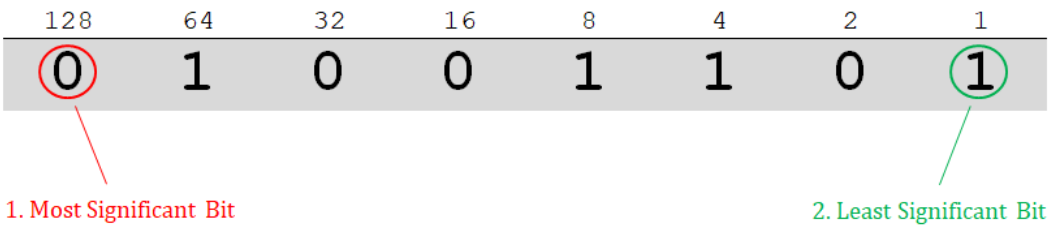


Figure 2.5: A labelled binary number representing the decimal number `77`.

MIDI files are essentially just long streams of bytes. These bytes are almost all used to represent small units of musical information known as *MIDI events* which are generally no more than a few bytes of data. There is a multitude of different events with different purposes, but they all contain basic information such as the time at which the event must fire, what type of event it is, and other parameters.

ANOMIC supports MIDI files for pattern annotation, due mainly to its widespread use and compatibility in the symbolic music domain, and in **Chapter 3** we will go into more detail on how such files were parsed. In the next section, we will discuss different methods of representing MIDI files in a human-readable interface.

2.2.3 Interfaces for Viewing MIDI Music

MIDI files are stored as a stream of bytes with a rigid structure defining how the data is to be parsed, but this is of course nowhere near readable or editable for humans. We require an interface that can take MIDI data and illustrate the music in front of the user. There are multiple interfaces that do this, and in this section, we will discuss two of them.

The first method is already familiar: sheet music. While sheet music was historically associated with physical papers, it remains the tried-and-true method for notating and viewing music, so software solutions that offer integration of sheet music and the MIDI format are popular and still widely used to this day. Most music notation software programs, such as ("Finale Notepad"), ("Avid - Sibelius"), and ("MuseScore") are able to convert sheet music notated through their interface into MIDI, and there also exists software that is able to convert MIDI into readable sheet music, such as ("Midi Sheet Music"). **Figure 2.6** shows an example of a digital interface that allows editing of sheet music and MIDI file support.

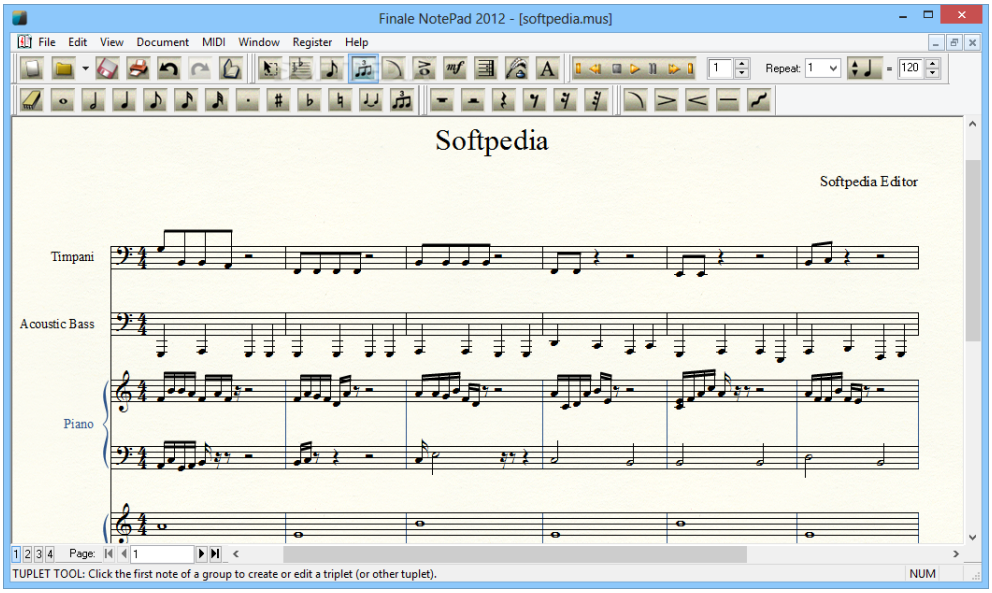


Figure 2.6: The interface of the music notation software Finale Notepad 2012.

While these interfaces are used in practice by many, there are some problems that arise when considering how compatible sheet music is with the MIDI format. Translating from sheet music to MIDI data is arguably not perfect; you lose various elements of sheet music such as performance directions. For instance, the word *rubato* is used as a performance direction in sheet music, and it states that the music to which it applies is to be performed with a loose and expressive tempo. The MIDI format does not specify a method of faithfully reconstructing this and many other performance directions, meaning that a lot of the artistic expression that notated sheet music offers is lost in the conversion.

It is potentially even more complex to perform the reverse process, which is to convert MIDI files to sheet music. Certain important elements of the music, such as fingering or which hand plays which note in a piano piece, are not going to be present in the MIDI file, and reconstructing the file in a readable and sensible manner is non-trivial. Furthermore, recall that MIDI events have a delta-time parameter associated with them, indicating which point in the music they are supposed to be executed. In sheet music, durations and times are quantised into bars, beats, and specific note durations (quarter notes, eighth notes, etc), but this delta-time parameter in MIDI events has no limitations on what value it can be. Depending on the origin of the MIDI file, it is certainly not guaranteed that all times precisely fit the pre-determined time values of sheet music (especially if, say, the MIDI file contained note data that was played by a human performer). This incompatibility motivated a need for a more apt interface for representing MIDI data in a human-readable setting: the piano roll interface.

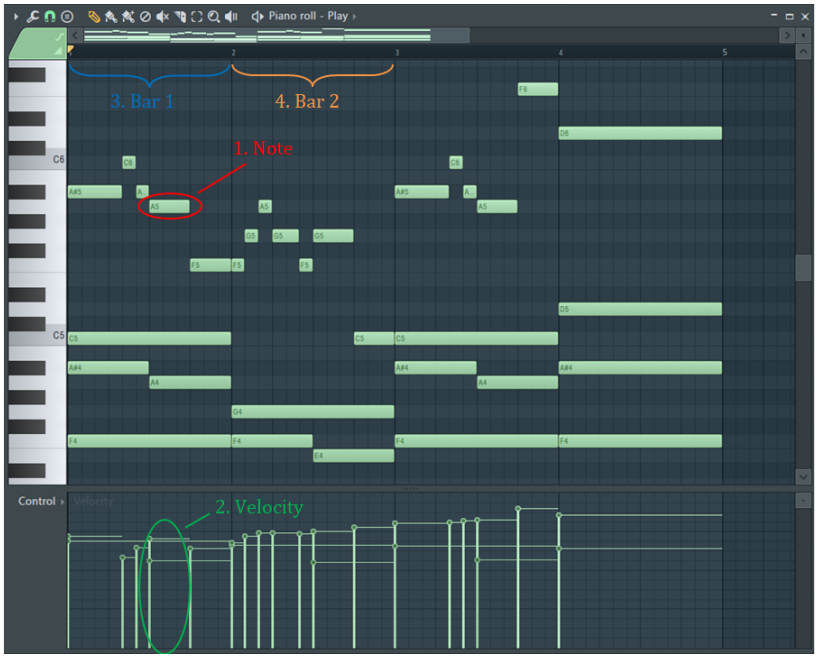


Figure 2.7: A labelled piano roll interface from the program FL Studio.

Figure 2.7 shows an example of the piano roll interface in use by a digital audio workstation software ("FL Studio"). It is based on an actual piano keyboard, with the piano keys visible on the left (the C notes are marked with a number near them indicating

their octave). The grid to the right of the piano keys showcases a simple melody with chordal accompaniment, and an example note (1) is labelled. The velocity of the note in question (2) is found below it, and higher values will lead to louder notes while lower values signify softer notes. Just like in sheet music, the grid containing the notes is split into bars, and two examples (bar 1 (3) and bar 2 (4)) have been labelled.

It is immediately clear that the piano roll interface is a more compatible representation of MIDI data. It is able to represent notes with the same time resolution as a MIDI file would, it supports note velocities, and it detaches itself from performance directions and other elements unique to sheet music in favour of being fully compatible with MIDI data. For this reason, it was the chosen interface for the developed annotation tool.

2.2.4 The JAMS Format

JAMS (Humphrey et al., 2014), which stands for JSON-Annotated Music Specification, is a JSON-based schema for use in music information retrieval research. JSON, which stands for JavaScript Object Notation, is a data-interchange format that is easy for both humans and machines to parse, which makes it a practical and popular format for saving objects to a text format. JAMS files are used to store music annotations, which can be anything from beat detection, chord recognition, or, in the case of our research, musical patterns. As it aims to be the standard for reproducible MIR research, it was chosen as the format for saving or loading annotations.

The best way to describe the JAMS annotation schema is by use of an example. Below is a snippet of the first few lines of a JAMS file:

```
{
  "sandbox": {},
  "file_metadata": {
    "duration": 28800,
    "title": "bach2.mid",
    "release": "",
    "identifiers": {},
    "artist": "",
    "jams_version": "0.3.3"
  },
```

The format is evidently following the format of standard JSON data, where curly brackets indicate the start or end of an object, square brackets indicate the start or end of an enumerable set of objects, string values or names of object attributes are encapsulated within quotation marks, the values of the attributes are specified to the right of the colon, and attribute name/value pairs are separated by commas.

The JAMS format specifies a list of attributes that are common to all JAMS files, and the first of these attributes specified in the example is the `sandbox` attribute. This is used to store data about the JAMS file that is not necessarily structured according to JAMS rules. The `file_metadata` attribute, on the other hand, establishes the metadata

of the file being annotated. In the example, it states that a MIDI file called bach2.mid, which had a duration of 28800, was annotated using a JAMS version of 0.3.3. The next snippet of the example describes the actual annotation data.

```
"annotations": [
    {
      "sandbox": {},
      "duration": 28800,
      "namespace": "pattern_jku",
      "time": 0,
      "annotation_metadata": {
        "corpus": "",
        "validation": "",
        "annotation_tools": "",
        "version": "1",
        "curator": {
          "name": "",
          "email": ""
        },
        "annotation_rules": "",
        "annotator": {},
        "data_source": ""
      }
```

The square bracket indicates that the `annotations` object is in fact an enumerable set of multiple collections of annotations. Once again, the `sandbox` attribute will contain unstructured data about the annotations in this collection. The `time` and `duration` attributes indicate the period of time in the music for which the annotations are valid. The `annotation_metadata` attribute contains metadata about the annotations.

Lastly, the `namespace` attribute indicates the type of annotations, which, in our case, is `pattern_jku`. This namespace defines the format that is based on pattern annotations from the JKU Patterns Development Database ("GitHub - JKUPDD-Aug2013"), which is a dataset that has repeated pattern occurrences annotated for use as a ground truth in pattern discovery research. The last snippet we will consider will showcase the format of a `pattern_jku` annotation.

```
 "data": [
        {
          "value": {
            "pattern_id": "0",
            "midi_pitch": "69",
            "occurrence_id": "0",
            "morph_pitch": "69",
            "staff": "0"
          },
          "confidence": 3,
          "time": 672,
          "duration": 95.8
        },
```

This is the first of many annotations listed in the file. JAMS annotations of any type must have a `confidence`, a `time`, and a `duration`. The `value` object has multiple different attributes that are unique to the *pattern_jku* namespace. Each annotation in this format represents a single note in the piece of music being annotated, and the `occurrence_id` indicates which pattern occurrence the note belongs to. The `pattern_id` attribute, on the other hand, specifies the ID of the pattern to which the pattern occurrence belongs.

There is an important distinction at play: From the perspective of the `pattern_jku` namespace and, indeed, this thesis, a `pattern` is an abstract set of notes, but the actual manifestations of that pattern in the music are defined to be its occurrences. A pattern with zero or one occurrences is not considered a pattern from our perspective, because, as mentioned in the previous chapter, patterns are built upon the idea of repetition. If a pattern occurrence does not repeat, exactly or otherwise, then it is not considered as representative of any pattern by our definition.

The other attributes are the `midi_pitch`, the `morph_pitch`, and the `staff`. The `staff` indicates which musical staff (or stave) in the sheet music the note belongs to, and it is used by our annotation tool as indicating which `MIDI channel`[3] the note belongs to. `midi_pitch` is a number from 0 to 127 that states the pitch of the note. This corresponds directly with the pitches of notes found in MIDI files. The `morph_pitch` stands for the *morphetic pitch* of the note (Meredith, 2006), and helps to indicate the position of the note in music notation. This attribute exists because of the phenomenon that one pitch in sheet music can have multiple representations. For instance, E flat (the E note but lowered a semitone) and D sharp (the D note but raised a semitone) are considered the same pitch, but depending on the key or tonality of the music, one or the other may be used when notated in sheet music. The morphetic pitch is therefore the indisputable position of the note on a musical stave. Understanding tonality and the morphetic pitch is beyond the scope of this research, and for all intents and purposes, the `midi_pitch` attribute will be the only relevant pitch attribute in this thesis.

2.3 Related Work

At the time of writing, there are no other digital MIDI annotation tools that have a similar interface and functionality to the one presented for this thesis, but there have been various initiatives that have the similar goal of gathering or facilitating musical annotation data.

In (Russell, Torralba, Murphy, & Freeman, 2008), a tool is presented with a similar concept of allowing humans to annotate content, with the exception that it operates in the images domain. The *LabelMe* tool aspires to provide a web-based platform for users to annotate objects inside images (for instance, annotating a tree with a bounding box and label in an image of a neighbourhood). User-submitted annotations are compiled to build a database for use in computer vision research, particularly in the area of object detection and recognition.

[3]MIDI channels are discussed in more detail in **Chapter 3**.

Moving back to the music domain, a tool that supports some form of music annotation is *Sonic Visualiser* (Cannam, Landone, & Sandler, 2010), a program designed around visualising music in the audio domain, targeted specifically at music information retrieval research. It offers a large array of different features, including an annotation tool that allows the user to create annotation layers for musical segments that can be marked on the audio waveform or spectrogram. While its functionality on audio files is markedly comprehensive, it lacks support for annotating symbolic music.

Music annotation need not be limited to specialised tools. Software that encourages and facilitates music annotation has also been implemented in the form of games. The motivation behind this approach is that manual annotation is ultimately time consuming, costly, and difficult to scale, while a game would offer the user an engaging platform to partake in such an exercise. While this sacrifices some of the expert-level functionality and professionalism of a formal annotation tool, the hope is that it would increase the popularity of this task.

Examples of gamification of music annotation include the work done in (Turnbull, Liu, Barrington, & Lanckriet, 2007). In this paper, they describe the developed game, dubbed the *Listen Game*, which was used in a two-week pilot study to gather annotations from human players that then served as training data for an SML (supervised multiclass labelling) model. They used the performance of the SML to evaluate the annotations and, by extension, the game itself. The goal of this multiplayer game was to query all players about a given piece of music and have them label it with a description from a set of predetermined descriptions. An example of a query would be "use of the song", which is asking the user how the song could be used or in which context it would be valuable to listen to it, and some example labels that users can pick for that query would be "intensely listening", "driving", and "getting ready to go out". Depending on how high the agreement on the labelling was among the players, a score would be given out. Along with evaluating the performance of the SML model, both qualitative and quantitative analyses were performed on the annotations.

The theme of encouraging multiple players to agree with each other continues in the game *TagATune* (Law, Von Ahn, Dannenberg, & Crawford, 2007). In this game, two participants are asked to listen to a number of pieces of music and partake in two rounds; an annotation round and a comparison round. The former round would prompt the players to label the individual piece of music they are listening to with a suitable description from a list of predetermined labels (such as "funky", "cheesy", etc). The latter round would ask players to compare different tunes with each other on grounds such as preference, similarity, and perception. In both rounds, agreement among players is, once again, rewarded with a higher score.

The next section will discuss the methodology undertaken for the implementation of the annotation tool and associated research.

Chapter 3

Method

3.1 Overview

To recapitulate the earlier chapters, the problem we are attempting to tackle is the lack of digital infrastructure for human musical pattern annotation. We have explored previous work in this vein and provided background on musical terminology, what a musical pattern is, different forms of digital music, and our chosen symbolic format for annotating patterns in this domain: MIDI. This next section will begin exploring the methodology employed in the development and evaluation of the software annotation tool ANOMIC. Firstly, there is a section on MIDI file parsing and how it was implemented from the ground up for the annotation tool. Following that is a detailed look into ANOMIC's interface and functionality. Next, details on the saving and loading of annotations using the JAMS format will be divulged. Finally, a description of the tool's evaluation through the user study can be found at the end of the chapter.

3.1.1 Resources

This section briefly lists the resources used for the implementation.

ANOMIC was written in C# and XAML (Extensible Application Markup Language) within a WPF (Windows Presentation Foundation) environment, using the Visual Studio 2017 IDE ("Visual Studio"). Everything was developed from scratch, with the exception of MIDI playback handled by the Midi-Dot-Net ("GitHub - jstnryan/midi-dot-net"). The MIDI parser was written with reference to The Sonic Spot's invaluable tutorial on the format ("The Sonic Spot - MIDI File Format"), and the JAMS parser was written with reference to the JAMS documentation ("JAMS Structure - jams 0.3.2 documentation"). At the time of writing, ANOMIC only supports the Windows operating system, and it has been tested thoroughly on Windows 10.

The user study was distributed by email, with the survey prepared using ("Google Forms"). Results were analysed using Microsoft Excel 2017 ("Microsoft Office") and C#

scripts written in Visual Studio 2017. The user study was distributed using ("MEGA") cloud storage, with instructions and user manuals for the study written in Microsoft Word 2017 and ("Google Docs"). An instructional video was made where instructions on how to use the tool were narrated while showing the explained features through a screen recording of the annotation tool being used. The footage was recorded with ("NVIDIA ShadowPlay") and edited with ("Movavi Video Editor"), with narration recorded and edited using ("Audacity")'s audio editing tools and a ("Blue - Yeti") USB microphone.

3.2 MIDI File Parsing

The format in which the music being annotated for this project was saved is the MIDI format. As such, a significant amount of work had to be done to interpret MIDI files and present their content in ANOMIC's interface. This section will detail the MIDI parser built from scratch that was used to load files into the annotation tool. We will begin by studying the structure of a MIDI file.

3.2.1 Structure

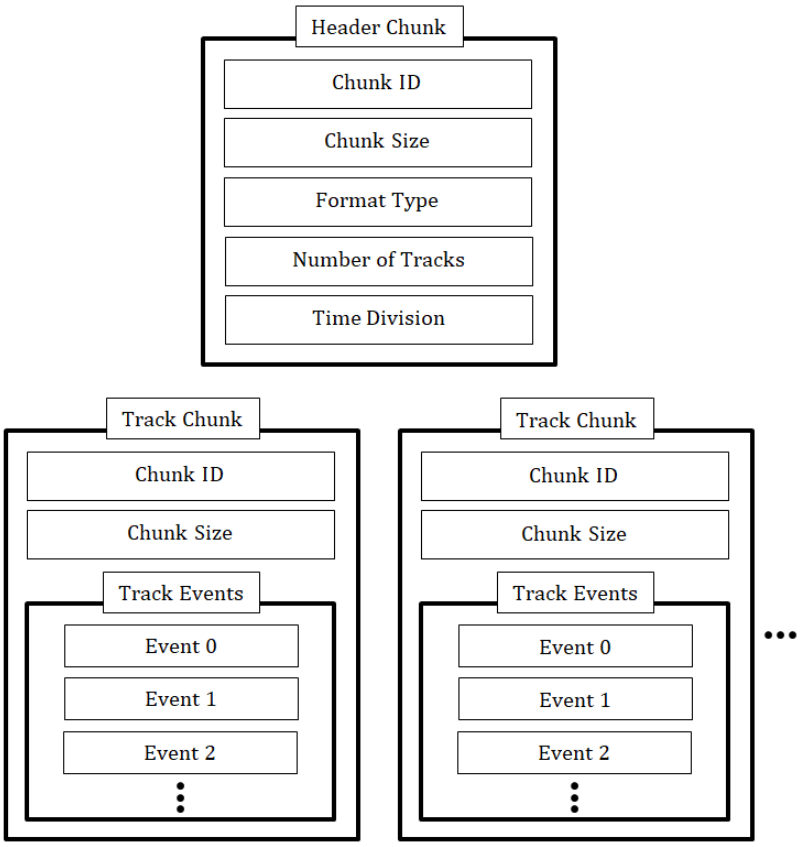


Figure 3.1: A diagram illustrating the structure of a MIDI file.

Figure 3.1 demonstrates the structure that a typical MIDI file will follow, and the next section will describe each constituent element in detail.

Header Chunk

The very first set of bytes that will be read correspond to the *Header Chunk*, which is a collection of basic, universal data about the MIDI file and its contents. Below are the elements that it is comprised of:

- The *Chunk ID* is simply an initial check done to ensure that the file being parsed is truly a MIDI file. It should contain a set of four bytes corresponding to the ASCII string `MThd`.
- The *Chunk Size* in a data chunk of a MIDI file defines how long to parse the rest of the chunk for. Since a header chunk always has a fixed length, the Chunk Size should always be `6`.
- The *Format Type* defines how the rest of the MIDI file will be formatted. It can be three possible values:
  - A value of `0` implies that there will be only one track chunk containing all the MIDI events of the entire song.
  - A value of `1` implies that there should be more than one track chunk in the file, with the first track, by convention, containing song information (tempo, time signature, etc), and the rest of the tracks containing music event data.
  - A value of `2` is similar to the type 1 format, except the tracks are not intended to be played simultaneously and instead are used to store various sequences of events such as drum patterns.
- The *Number of Tracks* value, as the name suggests, is the number of tracks (and, ergo, the number of track chunks) present in the MIDI file. If the header chunk specifies a Format Type of `0`, then this value will consequently be `1`.
- The *Time Division* value is a more complex piece of information that indicates how to interpret the time values of the MIDI data. The MSB (Most Significant Byte) of this two-byte value indicates the type of time division being employed in the MIDI file in question. With an MSB of `0`, the time division type is "PPQ" (Points Per Quarter) (Huber, 2007, p. 43), which is a format that states that the remaining fifteen bits of the Time Division value are the number of ticks per quarter note of the music. For example, a value of `00000000 01100000` has an MSB of `0`, implying the PPQ type. The remaining bits represent the decimal number `96`, which means that any time value that shows up in the MIDI file's events can be divided by `96` to obtain the time value in quarter notes. Conversely, if the MSB is `1`, then the time division type is "SMPTE" (Society of Motion Picture and Television Engineers), a standard time code format in use within many forms of digital media. Supporting

the SMPTE type was beyond the scope of the annotation tool, as the entirety of MIDI files used for testing and the user study followed the PPQ format.

After the Header Chunk, the chief part of the MIDI file can be found: the track data.

Track Chunks

The track chunks in the MIDI file contain all of the data that comprise the actual music. Depending on the header's Format Type value, there could be only one track chunk or multiple. Each track chunk will be formatted in the same manner, and here is a description of the different elements:

- Similarly to the header's variant, the *Chunk ID* identifies the track chunk with a four-byte string consisting of the letters `MTrk`.
- The *Track Size* defines how large in bytes the rest of the track is. This can vary wildly depending on how many events the MIDI file has.
- The *Track Events* are the bulk of the data. Each set of events in a track chunk can contain any number of events and it should continue to be parsed until the number of bytes in the Track Size value has been reached.

Finally, we will briefly discuss the format of MIDI events.

Track Events

The track events in a track chunk start from the beginning of the music and fire in a sequence until the track's chunk size has been reached. While events can come in many different types, they all abide to the following fundamental format:

- The *Delta-Time* is the first argument of any event, which determines when an event should fire relative to the previous event. This parameter is what is affected by the aforementioned Time Division value. For instance, assuming once again a Time Division of `96`, a *Note On* event with a delta-time of 384 implies that it should fire exactly four quarter notes worth of time after the previous event listed in the MIDI data. Unlike most data in MIDI files, the size of this element is *variable-length*, rather than predetermined.[1]
- The *Event Type* defines the type of musical information that is displayed. The vast majority of these events will be *Note On* and *Note Off* events, but other types of events can be used to signify the track's tempo (or tempo changes), time signature, key signature, and so on.
- The remaining data is simply parameters related to the type of the event. For example, the parameters for a Time Signature event determine what the time signature of the music is. The parameters for the Note On and Note Off events are pitch and

[1]More information on how variable-length data was handled can be found in **Section 3.2.2**.

velocity; the pitch parameter states the pitch of the note, and the velocity parameter states how loud the note is, velocity being the MIDI format's primary method of describing the dynamics of the music.

While it is out of scope of this section to go over all of the possible different types of MIDI events, there are some points to discuss regarding event types. An event type can fall under one of three broad categories: *Channel Events*, *Meta Events*, or *SysEx Events*.

Channel events are what comprise most of the musical information. They have a specific time at which they are meant to fire, and any kind of real-time musical data such as musical notes, MIDI control state changes[2], and pitch bends would be classified as channel events. The factor that makes this type of event unique is the fact that a channel event will always have a *MIDI channel number* associated with it. MIDI channels are used to differentiate between different MIDI inputs and can be used to segregate the musical data in a MIDI file.

Any other type of event would not have a channel number associated with it and would instead be a *Meta Event* or *SysEx Event*. Meta events would contain song information that is separate from the musical data needed to play the MIDI. Time/key signature events, lyrics events, instrument name events, and tempo events, are all examples of meta events. On the other hand, SysEx events, which stand for system-exclusive events, are the MIDI format's method of extending its functionality and customisability. Any event type that falls under this category will be for specifying manufacturer-specific events, so that any type of brand-specific MIDI event that is not necessarily supported by the base MIDI event types can still be represented using these SysEx events.

The next section will describe how the implemented MIDI parser was able to extrapolate events from MIDI.

3.2.2 Parsing to Events

The initial step of the MIDI parsing pipeline is to translate the stream of bytes into a list of MIDI events. This parsing process necessitates close regard to the MIDI format specification so that the correct number of bytes are parsed for each individual element of the MIDI file. **Figure 3.2** shows a flowchart of the algorithm used to parse a given MIDI file.

As the figure illustrates, all of the individual elements of the header chunk discussed in **Section 3.2.1** are read. To ensure that the file being read is indeed a non-corrupt MIDI file, it is ascertained that the header chunk ID and size are the correct values. The format type and track number are then read; recall that the format type indicates the structure of MIDI tracks that is present in the file, and the track number indicates how many track chunks are present in the file. Finally, the time division is read and its MSB

[2] A MIDI control is a unique parameter used to adjust certain aspects of a particular MIDI instrument such as portamento or panning.

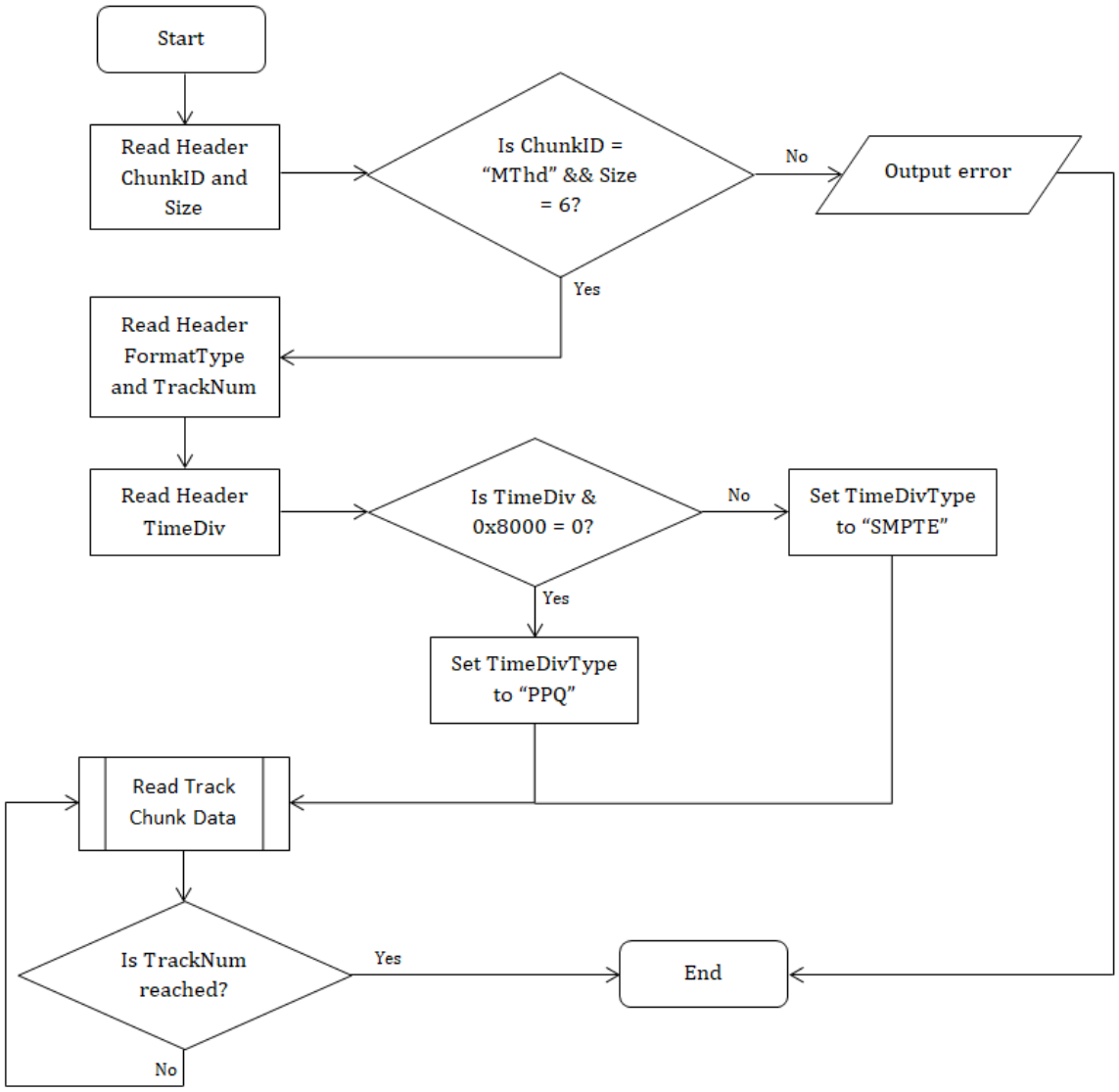


Figure 3.2: A flowchart of the algorithm used to parse the MIDI file header chunk.

is checked using a binary mask operation[3], assigning the correct time division type based on whether the MSB is `0` or `1`.

Now, the track chunk data must be parsed. Depending on the format type, there might be one or multiple track chunks, but there will always be at least one. For each track chunk, the algorithm illustrated by the flowchart in **Figure 3.3** was implemented.

As the figure demonstrates, the initial parsing done for the track chunk is similar to that done for the header chunk, which is to read the chunk ID and size, followed by verifying that the chunk ID is the correct value. Once that is parsed, the events immediately follow. As discussed in **Section 3.2.1**, the first argument is the delta time, or how long (in PPQ or SMPTE format) the event should take to fire relative to the previous event.

[3]A binary mask operation is a bitwise operation using logical operators such as AND, OR, and XOR that is often used to query the status of certain bits or return a subset of bits from a set of bits.

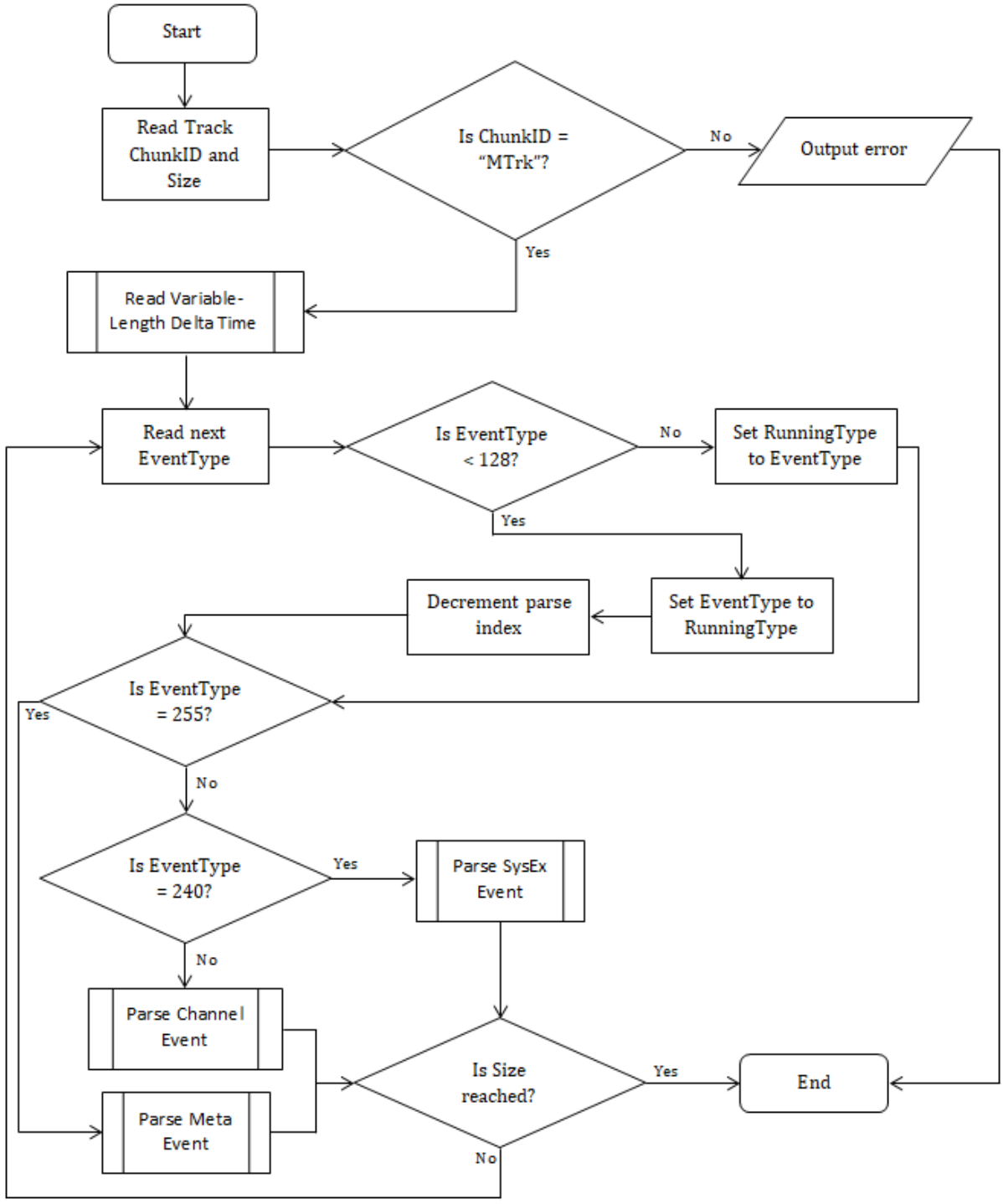


Figure 3.3: A flowchart of the algorithm used to parse a MIDI file track chunk.

The first hurdle to overcome is that this argument has a variable length, unlike most other parameters in a MIDI file which have known sizes. While the maximum size is four bytes, the actual size can be anything below that size as well. The reason behind this delta time parameter being dynamic pertains to the origin of the MIDI format; recall that MIDI was originally intended as a format for communication between music hardware. This communication had to be lightweight and quick, which meant cutting down on as many bytes as possible and only using what is needed.

The way MIDI parsers know when to stop parsing a variable-length value is by utilising the MSB of the byte being read by the parser to signal whether or not to stop parsing and then using the remaining seven bits for the value of the parameter. If the MSB is a `0`, then the byte being parsed is the last byte of the parameter. Otherwise, the next byte is also considered part of the parameter. For example, to represent the decimal number `128`, we would use the byte `10000000`. However, the MSB is treated as a status bit for variable-length data, so the actual variable-length data representation of `128` would need to carry over the `1` bit to another byte, making the result `10000001 00000000`. The `1` in the MSB of the first byte indicates that the next byte is part of the parameter, and the `0` in the MSB of the second byte indicates that this is the last byte in the sequence. What follows is pseudocode of the algorithm used to read such variable-length data:

```
byte[] GetVariableLengthData(index)
    oldIndex = index // save the parse index to a temporary variable

    // while current byte's MSB is not 0
    while midiFile[index] & 0x80 != 0
        index++

    offset = index - oldIndex

    // save all bytes of the argument to a byte array
    for counter = 0 to offset
        data[counter] = midiFile[oldIndex + counter]

    return data
```

The pseudocode is a function that takes the current index of the byte being parsed in the MIDI file, and returns all of the bytes that make up the variable-length data parameter being checked. These bytes are then amalgamated together in the code, the MSB of each byte is removed, and the final binary number is stored as an integer.

Once the delta time parameter is handled, the event type of the event currently being parsed comes next. Recall that this indicates what kind of musical data the event is comprised of and, most importantly, specifies how it should be parsed. As previously covered, the event type can fall under one of three classifications: a channel event, a meta event, or a SysEx event. That being said, before identifying the event type, a check is first done to determine whether the event type value being read is less than `128`. In reality, an event type can never be under this number, because meta events always have an event type of `255`, SysEx events always have an event type of `240`, and channel events store their event type in the most significant four bits and the channel number in the least significant four bits; since channel event types can never be `0`, this means that the entire byte should never be under 128.

The reason for this check is that MIDI files introduce a method of making the data even more lightweight and easily transportable: *Running Mode* (Huber, 2007, p. 46). This is a fairly simple protocol that states that if multiple events of the same type follow each other, then the event type value of the first event in this sequence can be assumed for the

remaining events, allowing the event type byte to be dropped for a large number of MIDI events. All parameters of data past the event type will be in the range of `0-127`, so when a byte is encountered that has a value that's greater than or equal to `128`, then it can be assumed that the new event type byte is to be parsed again.

After the event type has been established, the event's data is parsed. Depending on whether the event is a channel, meta, or SysEx event, and depending further on the specific type of event, the particular method of parsing and interpreting event data might vary slightly across different types. It is beyond the scope of this paper to divulge the precise parsing details of the different event types, but more details can be found on the SonicSpot format specification webpage ("The Sonic Spot - MIDI File Format").

Once the events have been parsed, we are left with a list of many different events of various types, most of them being Note On and Note Off events. The next step is to turn all of that into music.

3.2.3 Parsing to Music

This stage of the parsing implementation pipeline will iterate through the long list of events returned by the event parser and generate a set of musical notes, each having a pitch, duration, velocity, and MIDI channel number. This is not a simple task, since MIDI events corresponding to musical notes are saved as Note On and Note Off events, with the former being an indication that the note should start sounding and the latter being an indication that the note should stop sounding.

Turning this collection of events into notes was done by storing a dictionary for "active notes", with the dictionary key being note pitch and the dictionary value being a stack of notes that possess that pitch. Every time a Note On event that had a particular pitch was found in the list of events, it would be saved in the dictionary entry corresponding to its pitch at the top of the note stack. When a Note Off event is found, the algorithm would go to the dictionary entry corresponding to this event's pitch, and pop the last note off the stack. This note's duration, channel, pitch, and velocity would be calculated, and the note object would then be saved. Below is pseudocode showing the above process:

```
timeElapsed = 0

for each midiEvent
    timeElapsed += midiEvent.Time

    // if the current event is Note On, add it to the dictionary stack
    if midiEvent.Type = NoteOn
        activeNotes[midiEvent.Pitch].Push(midiEvent)

    // if the current event is Note Off, pop its Note On counterpart from the stack
    if midiEvent.Type = NoteOff
        newNote = activeNotes[midiEvent.Pitch].Pop()
        newNote.Duration = timeElapsed - midiEvent.Time
        notes.Add(newNote)
```

This approach yields a complete list of all of the notes present in the MIDI file, and most importantly it is robust to polyphony. Saving a dictionary of active notes allow many to be active at any one time and only the correct Note On events are "terminated" by a Note Off event when it appears.

Notes alone are not the only musical information that needs to be extracted from the events. The two other event types that were considered for the music parser were meta events: *Set Tempo* events and *Time Signature* events[4]. These events, in a type 1 format of MIDI file, will often be in the very first track chunk, will not have any delta time associated with them, and will contain parameters that are used to calculate the tempo (in BPM) and the time signature of the music.

The Set Tempo event comes with one three-byte parameter, Microseconds/Quarter-Note ($MPQN$) and one can calculate the tempo from it using the following formula:

$$Tempo = \frac{60000000}{MPQN}$$

The Time Signature event comes with four one-byte parameters: the numerator, denominator, metronome pulse, and number of 32nd notes per MIDI quarter note. The former two parameters are the relevant values for extrapolating the time signature. The numerator parameter defines the top number of the time signature, while the denominator parameter specifies the value to which the power of 2 must be raised to equal the bottom number of the time signature. For instance, a time signature of `6/8` would have a numerator parameter of `6` and a denominator parameter of `3` (because 2^3 is `8`).

Having a list of all of the notes (with their respective pitches, durations, channel numbers, and velocities), the time signature, and the tempo gives us the means of outputting the notes on a piano roll. **Figure 3.4** shows an early screenshot of ANOMIC during development, simply plotting the notes with their pitches as rectangles on a plain white space. As with the previous example of a piano roll shown in **Section 2.2.3**, the rectangles' vertical position will indicate their pitch while the horizontal position will indicate their time in the music. The length of the rectangle states how long the note will sound for.

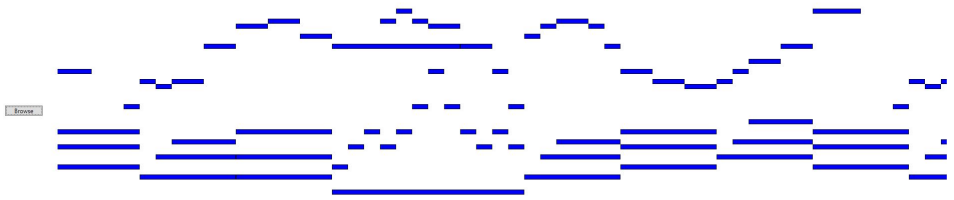


Figure 3.4: A very early prototype of ANOMIC's piano roll.

[4]In the absence of these events, a tempo of `120` BPM and time signature of `4/4` is assumed.

3.3 Interface

This section deals with all of the interactive and visual elements of ANOMIC.

After running the annotation tool, the first screen that appears is an extremely simple welcome screen (see **Figure 3.5**). There is a *File* menu at the top left where the user can load a MIDI file or exit the software. Alternatively, clicking the *Browse* button underneath the graphic will also allow the user to browse for a MIDI file using the standard Windows file browsing plug-in.

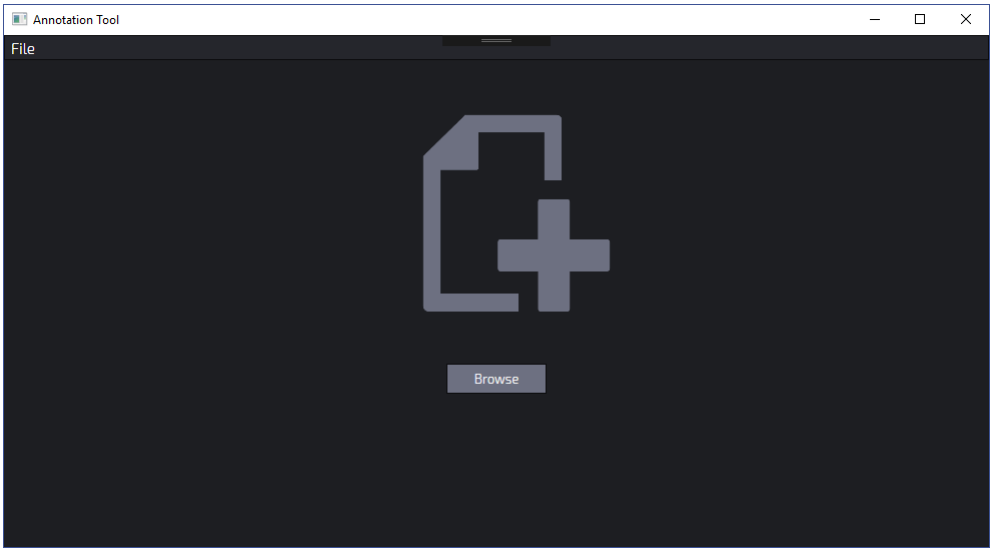


Figure 3.5: The first screen that appears after loading the annotation tool, ANOMIC.

Once the MIDI file is chosen, it will undergo the parsing process outlined in **Section 3.2**. Upon completion, ANOMIC will transition to a new screen, which is the main interface of the software. In this screen, all of the tool's functionality will become available to the user. This interface can be seen in **Figure 3.6**.

At the top of the interface, there is a conventional set of menu buttons that encompass most of the features that the tool supports. The below points list each menu button and summarise the selections present within each:

- The *File* menu contains all the functionality related to the input and output of files. More information can be found in **Section 3.4.2**.
- The *View* menu contains functionality related to adjusting visual elements of the tool. This menu is where the user can set horizontal or vertical zoom factors for the piano roll, show or hide occurrence graphics, show or hide the piano roll grid, and show or hide key names on the piano keys. More information can be found in **Section 3.3.1**.
- The *Patterns* menu deals with the addition and deletion of patterns. More information can be found in **Section 3.3.2**.

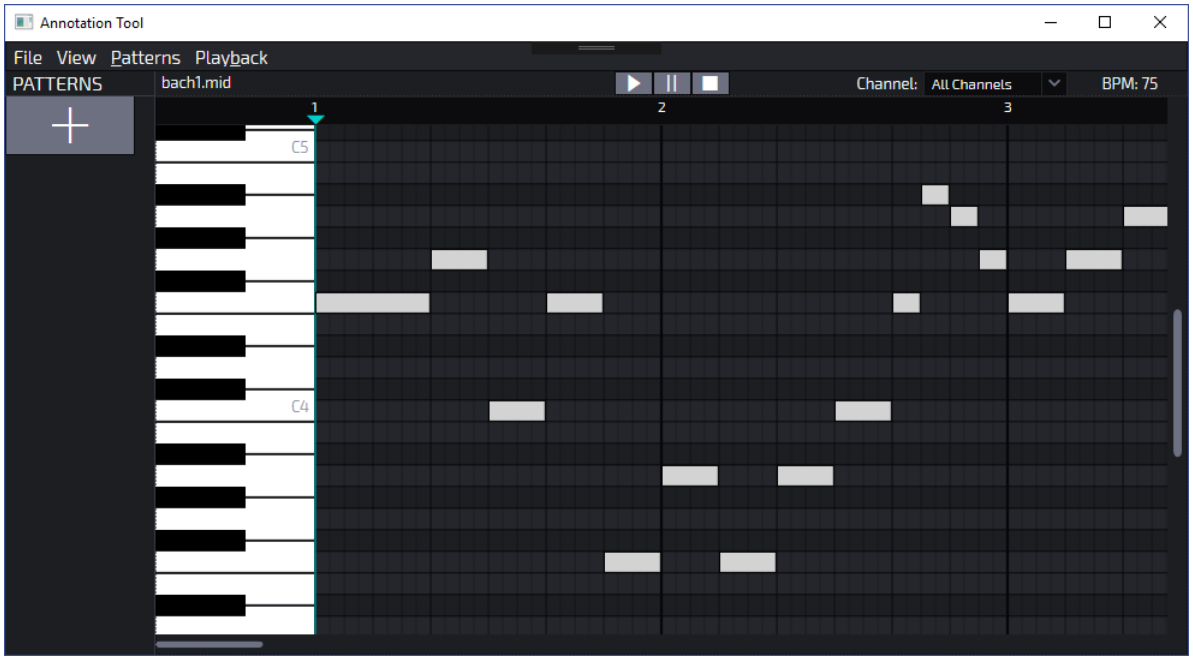


Figure 3.6: The main interface of ANOMIC.

- The *Playback* menu provides the user with some features and options related to playing the music back. More information can be found in **Section 3.4.1**.

The next sections will go over the different elements of the interface and functionality of the tool, describing their implementation.

3.3.1 Piano Roll

The piano roll is the section of the interface concerned with facilitating viewing of the MIDI file. **Figure 3.7** is a labelled screenshot showing ANOMIC's piano roll after loading an excerpt of a Bach cantata (Bach, 1725).

The grid (1) was implemented using shaded rows that coincide with piano keys and vertical lines that vary in thickness which coincide with different subdivisions of time in the music (the most notable subdivision being bars which have the thickest lines and bar numbers above them). This grid is where the notes parsed from the MIDI file can be found, and the visibility of these grid lines can be toggled from the View menu.

To the left of the grid is the set of piano notes (2) laid out vertically to help the user identify which pitch each of the notes in the grid corresponds to. There is text on each C note, with the number stating the octave of the key. The View menu provides options to disable the visibility of these key names or alternatively to even show key names for every single piano key, making the tool more accessible to users who have never played piano before.

The playback buttons (3) can be found at the top of the piano roll interface, allowing the user to play, pause, and stop the music. To the right of the playback buttons is the

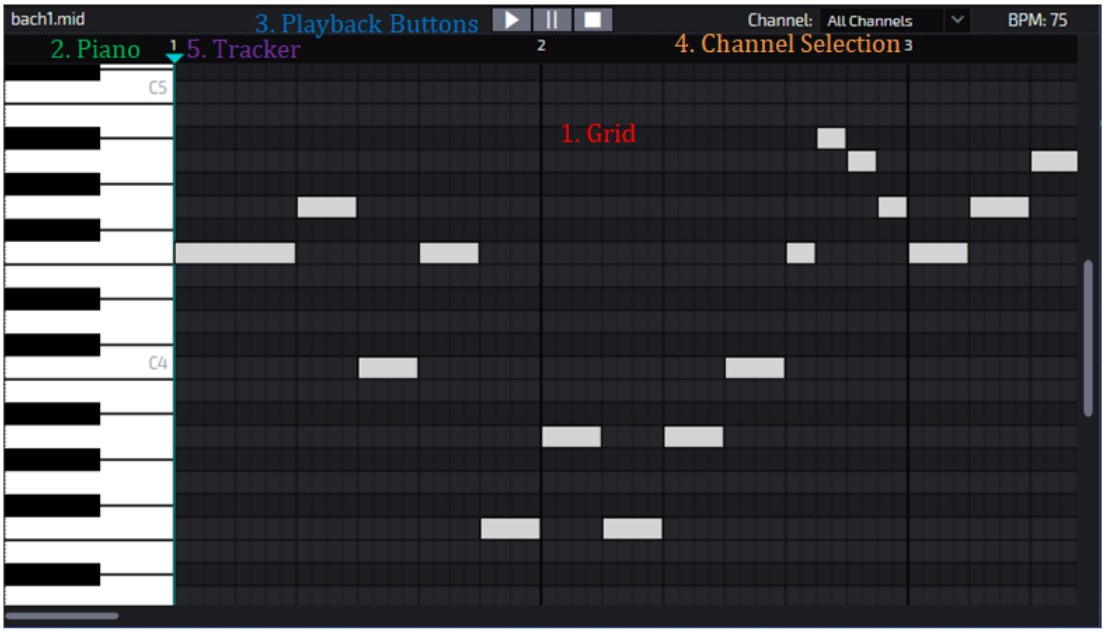


Figure 3.7: A labelled screenshot of the piano roll interface of ANOMIC.

channel selection menu (4). This is simply a drop-down menu that toggles the visibility of notes in the grid based on what channel they are in. Recall that every note event in the MIDI file will have a channel number, and MIDI channels are ways of differentiating between different sets of notes. The user can opt to see all of the notes in the grid or else to show notes that belong to only one of the channels. Finally, there is a tracker that is positioned at the very start of the music in the figure. This vertical line will move left and right to represent the current playback position of the music. The user can also drag it to manually adjust the playback position.

3.3.2 Patterns & Occurrences

To the left of the piano roll is the pattern panel, which deals with the addition, removal, and viewing of patterns and occurrences. **Figure 3.8** shows this panel with an example set of patterns and occurrences already prepared.

From the figure, one can see that there are three different patterns, each marked with a number and colour-coded. Under each pattern icon is a list of occurrences that belong to that pattern, also numerically named. Adding a new pattern for a MIDI file involves clicking on the large `+` button at the bottom of the panel. In addition, the tool constantly requires that the user has a pattern selected (by clicking on its respective icon in the pattern panel). This makes it impossible to annotate an occurrence unless it is associated with a pattern.

In **Figure 3.8**, the first and second pattern have two and three occurrences respectively, and the currently-selected pattern is pattern `3` with no occurrences. With this pattern selected, clicking anywhere on the piano roll grid will allow the user to begin selecting notes to add to pattern `3`'s first occurrence. Clicking on notes selects or deselects

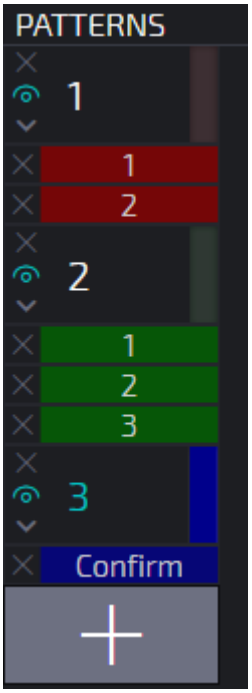


Figure 3.8: A screenshot of the pattern panel with three patterns created, each with a different number of occurrences.

them, and the user can also opt to click and drag their mouse cursor to select notes in bulk. This operation can be seen in **Figure 3.9**. After the notes are selected, the user can press *Confirm* on the pattern panel to add the occurrence and highlight its notes.

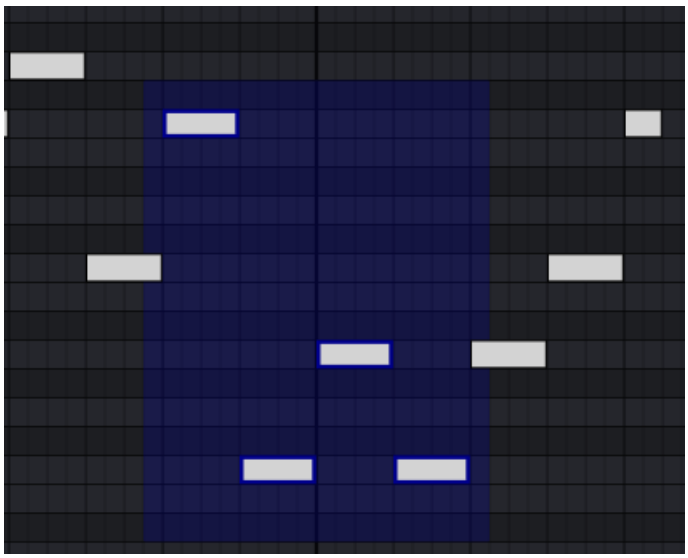


Figure 3.9: A screenshot of the user selecting four notes using a click-and-drag mechanism to add to an occurrence of the third pattern.

Figure 3.8 also shows some additional functionality related to the pattern panel. Each occurrence icon has a delete button to the left of it, and each pattern icon has three distinct buttons: a delete button, a view toggle button, and an expand/collapse toggle

button. The delete buttons are, of course, to delete the particular pattern or occurrence that they are associated to. The view toggle button allows the user to toggle the visibility of the occurrences of the associated pattern in the grid. Lastly, the expand/collapse toggle button will expand or collapse the occurrences directly beneath the associated pattern, making it easier to navigate the pattern panel when many patterns and occurrences are present. **Figure 3.10** shows how the piano roll would look after selecting notes for three occurrences, each from three different patterns.

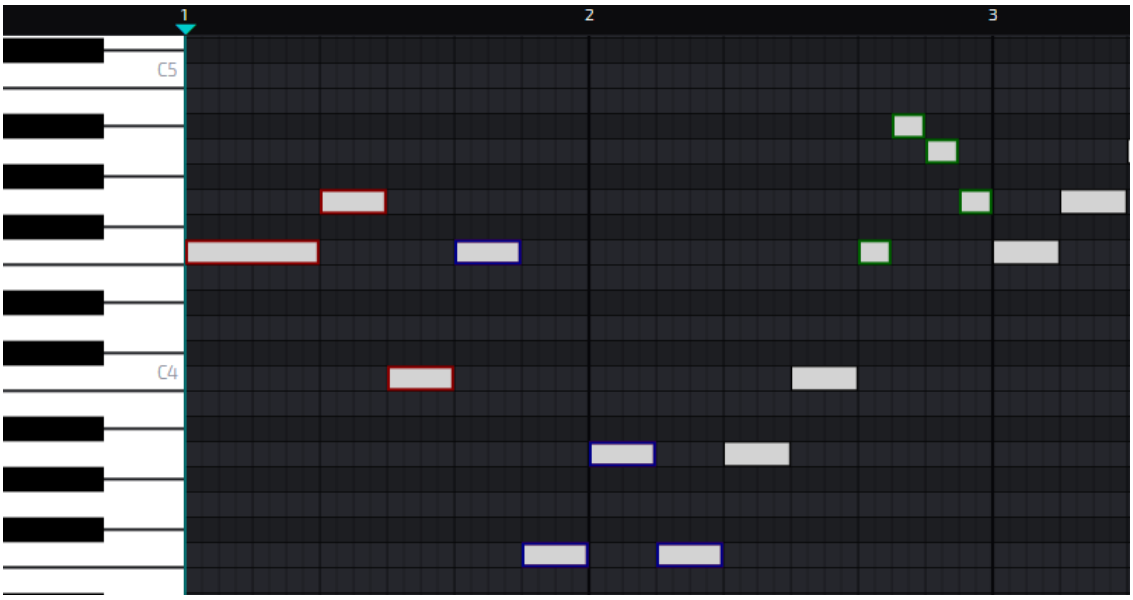


Figure 3.10: A screenshot of the piano roll after occurrences have been selected and each occurrence's notes have been highlighted.

With most of ANOMIC's interface described and illustrated, the next section will describe the implementation details of particular features of the tool.

3.4 Functionality

This section will share implementation details on functionality present within ANOMIC, namely how playback was handled, how file I/O works, how the automatic occurrence finder works, and how user actions were logged to file for the purposes of the user study.

3.4.1 Playback

Playback refers to the operation of starting, stopping, and pausing the music. As mentioned in **Section 1.4**, one advantage of moving musical annotation to a digital tool is the ability to listen to the music being annotated in real-time. Being able to listen to music allows easier identification of patterns compared to just looking at a score or piano roll, and therefore this was deemed a high-priority feature.

Midi-Dot-Net ("GitHub - jstnryan/midi-dot-net") was the library used to play the MIDI notes. It was chosen for its ease of use and adequate functionality. This library invokes Win32 API ("C++ Programming/Code/API/Win32") to access the MIDI sound module of the user's machine to play MIDI notes, and it wraps this functionality in an object-oriented API. This means that it comes bundled with various classes and class methods to support MIDI playback. **Figure 3.11** describes the process for MIDI note playback using this library.

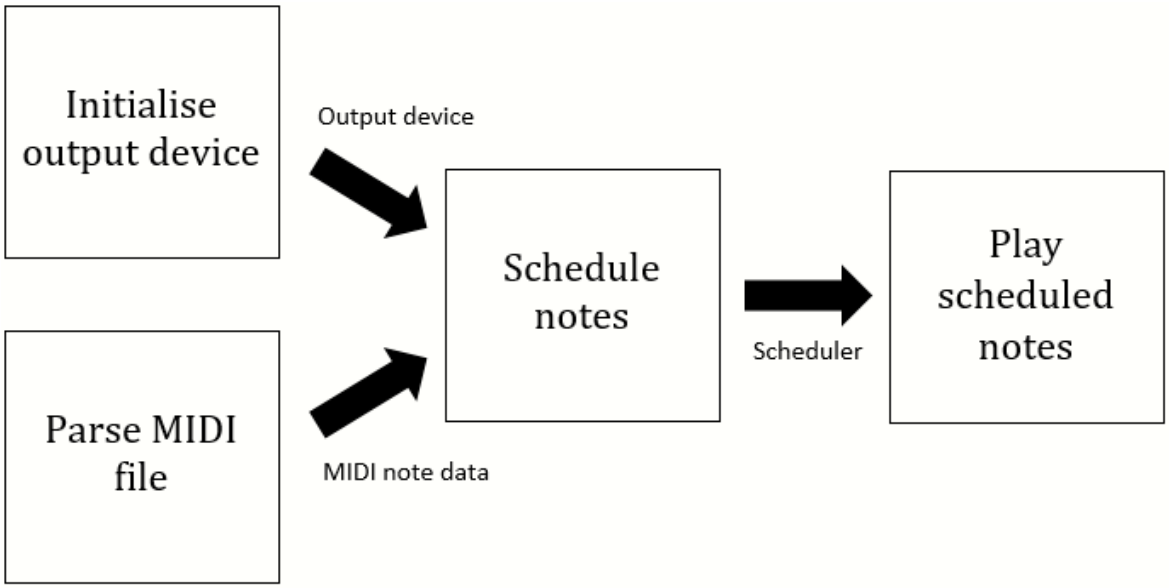


Figure 3.11: A diagram illustrating the pipeline for playing MIDI files.

The first two steps in the pipeline are to initialise the output device and parse the MIDI file. The process of the latter is already covered in **Section 3.2**, and the process of the former involves using the Midi-Dot-Net library to open the MIDI sound module of the user's machine. This module, referred to here as the output device, is concerned with audibly playing the MIDI notes. A reference to the output device, along with all of the MIDI information produced by the MIDI parser, is fed into a scheduler, which is another class from the Midi-Dot-Net library. The scheduler's job is to schedule all of the MIDI notes received from the parser in such a way that each note played by the output device is sounded at the correct time, in the correct MIDI channel, at the right pitch, and with the associated note velocity. With all of the notes scheduled, the piece can be played by calling `scheduler.Play()` whenever the user presses the play button.

This procedure becomes somewhat more complex when introducing pausing, stopping, and a tracker that can be manually moved and repositioned by the user. As covered earlier, the annotation tool features a tracker that indicates precisely which point of the MIDI file is being played in real-time. Every time the tracker goes over a note (i.e. that note is currently being sounded), the note is highlighted. The tracker and note highlights can be seen in **Figure 3.12**.

Figure 3.12: A screenshot of the tool showcasing the playback tracking functionality.

Due to the user's ability to pause the music at any point and reposition the tracker, the scheduler would need to update dynamically every time the user interacts with the playback position in any way. For example, if the user pauses the music precisely after the first three notes of a piece have fully played, then when pressing play again, the music should start from the fourth note, not the start. This means that the scheduler needs to be updated by scheduling all of the notes *except* the first three in anticipation of when the user presses the play button next time. Likewise, this scheduler update needs to be done whenever the tracker is manually moved.

Due to the fact that the Midi-Dot-Net library is quite basic, there are some limitations in the playback, with the largest drawback being that the only MIDI instruments that the library supports are piano and percussion. This entails that even if the annotation tool is fed a MIDI file with a variety of different instruments playing the notes (such as an entire orchestra), the playback will default all of the sounds to the MIDI sound module's piano sound (or the percussion sound if played in the MIDI channel designated for percussion).

3.4.2 Saving and Loading Files

As we want to save and load annotations during each session with ANOMIC, we require the permanence of the file system and the ability to generate and parse JAMS data. This section deals with how this feature was developed for the annotation tool.

Starting with file output, the version of ANOMIC distributed to participants for this thesis comes with user action logging. Every action performed by the user using the tool, along with the time at which it was performed and any unique parameters, is logged in the `sandbox` value of every `.jams` file that is saved by the user. Below, **Table 3.1** depicts all of the possible actions that are logged to file with each action's unique parameters:

Action	**Parameters**	**Description**
Session	SessionType	Starting or ending a session.
LoadedFile	FileName	Loading a `.midi` file.
PatternImport	FileName	Loading a `.jams` file.
SnapUpdate	SnapValue	Updating grid snap setting.
HorizZoomUpdate	ZoomValue	Updating horizontal zoom.
VertiZoomUpdate	ZoomValue	Updating vertical zoom.
KeyVisibilityUpdate	KeyVisibility	Updating piano key name visibility.
GridVisibilityUpdate	GridVisibility	Updating grid line visibility.
AutomaticIconsUpdate	IconVisibility	Updating visibility of the icon that appears on any occurrences found automatically.
ExpandAll	None	Pressing on the "Expand all occurrences" menu option.
CollapseAll	None	Pressing on the "Collapse all occurrences" menu option.
ShowAll	None	Pressing on the "Show all occurrences" menu option.
HideAll	None	Pressing on the "Hide all occurrences" menu option.
Play	None	Playing the MIDI file.
Pause	None	Pausing the MIDI file.
Stop	None	Stopping the MIDI file.
NormaliseVelocitiesUpdate	NormaliseMode	Pressing the menu option to normalise all velocities to 127.
ChannelUpdate	Channel	Updating the MIDI channel being viewed.
PatternAdd	PatternNumber	Adding a new pattern.
OccurrenceAdd	PatternNumber, OccurrNumber	Adding a new occurrence to a pattern.
PatternViewUpdate	PatternNumber	Toggling visibility of a pattern.
PatternSoloUpdate	PatternNumber	Toggling solo view of a pattern.
PatternExpand	PatternNumber	Expanding a pattern's occurrences.
PatternCollapse	PatternNumber	Collapsing a pattern's occurrences.
OccurrenceFindSimilar	PatternNumber, OccurrNumber	Utilising the automatic occurrence matcher for a particuar occurrence.
OccurrenceConfidenceUpdate	PatternNumber, OccurrNumber	Updating the level of confidence of an occurrence.
PatternDelete	PatternNumber	Deleting a pattern.

Table 3.1: A table of all possible user actions that are logged alongside the parameters logged with each action.

These logs are saved in memory as the user uses the tool, and when the menu option to save annotations is clicked, a new text-based `.jams` file is created. The first set of data is the `sandbox` JAMS parameter, which was used to store the above user logs. This is followed by the metadata of the `.jams` file, which is basic data such as the JAMS version and file title. Finally there is the annotation data. The JAMS namespace used for the annotations created in this study is the `pattern_jku` namespace, which translates well to the annotation data saved by the tool. The following is pseudocode for the algorithm that translates the patterns and occurrences saved by the tool into `.jams` annotations:

```
// initialising list of pattern_jku annotation objects
List<jamsAnnotation> annotations

for each pattern
    for each occurrence of pattern
        for each note of occurrence
            // saving pattern data from the tool to a .jams-friendly annotation
            tempAnnotation.time = note.time
            tempAnnotation.duration = note.duration
            tempAnnotation.confidence = occurrence.confidence
            tempAnnotation.pattern_id = pattern.index
            tempAnnotation.occurrence_id = occurrence.index

            annotations.add(tempAnnotation)
```

The above code generates a list of annotations that contain the parameters used in the `pattern_jku` namespace. In the `.jams` file, after specifying annotation metadata (such as the time and duration during which the annotations are applicable or the namespace of the annotations), these annotations are listed.

File input is, of course, the reverse process. ANOMIC accepts `.jams` annotation files, and it loads them by first requiring the user to load the MIDI file that the annotations are associated with and then browsing for the `.jams` file. As is apparent from the above pseudocode, the annotations are saved note by note, so when importing `.jams` annotations, the program has to parse through the text, load all the `.jams` annotations note by note, and then iterate through that list of note annotations to form a list of patterns and occurrences to save to memory and visualise on the interface.

3.4.3 Automatic Occurrence Matching

Automatic occurrence matching is a feature of ANOMIC that adds some quality of life and takes advantage of the ability to automate certain menial processes of music annotation. This section will briefly introduce the feature and go over how it was developed.

As previously established, we are concerned with repeated segments of music. These segments can be exactly or inexactly repeating, and one of the non-trivial steps of the process of finding patterns is choosing an initial set of notes that constitutes as a musically significant pattern. Finding similar occurrences after that is, of course, another challenge entirely, but if we make the assumption that there are occurrences of that initial set of

notes that repeat *exactly*, then we can automate the process of discovering these exact repetitions of an initial set of notes with an algorithm.

This is what the automatic occurrence matcher in the annotation tool aims to do. At any point during the session, the user can right-click on an occurrence that they had already found and initiate the automatic occurrence matcher for that occurrence. It will comb through the entire track and find occurrences that are exact repetitions of the occurrence in question, even if these repetitions may not be in the same time or pitch position. This implies that it is transposition-invariant and also works through polyphony.

Figure 3.13 showcases the interface in the patterns panel on the left when using the automatic occurrence matcher. The user creates the first occurrence and then right-clicks on it as seen in image (a). Then, the tool finds five other transposition-invariant, polyphony-robust occurrences that repeat exactly and marks those on the interface as being automatic with an icon near each occurrence. This can be seen in image (b), where occurrences 2-6 of pattern 1 are occurrences that repeat the sequence of notes of occurrence 1 exactly (aside from potential transposition).

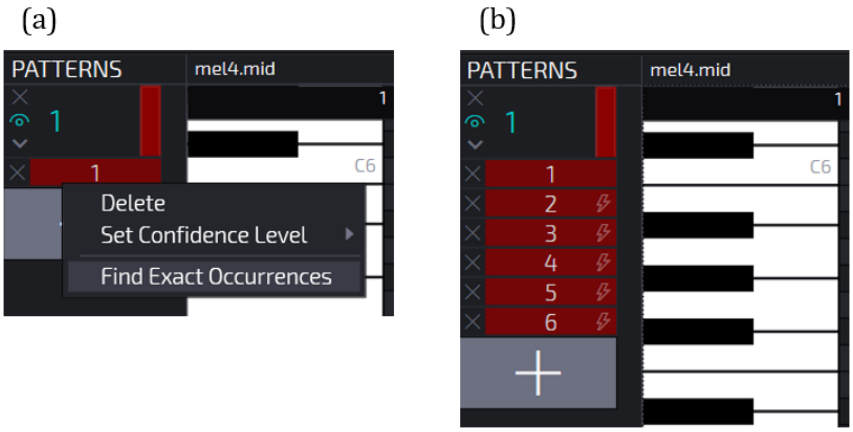


Figure 3.13: Two screenshots of the tool showcasing the automatic occurrence matcher functionality.

Below is the pseudocode for the algorithm that is in charge of retrieving exact repetitions of a given occurrence.

```
List<Note> tempNotes
bool found = false

for each midiNote
    // if the current note matches the duration of the occurrence's first note
    if midiNote.duration = occurrence.notes[0].duration
        tempNotes.Add(midiNote)

        // iterating through the occurrence's notes
        for i = 1; i < occurrence.notes.size; i++
            currentNote = occurrence.notes[i]
            previousNote = occurrence.notes[i-1]
            nextNotes = GetNotesAtTime(midiNote.time + currentNote.time
```

```
            - previousNote.time) // robust to polyphony
        found = false

        for each nextNote in nextNotes
            // transposition invariance
            if ((nextNote.pitch - midiNote.pitch =
              currentNote.pitch - previousNote.pitch)
              && (nextNote.duration == currentNote.duration))
                tempNotes.Add(nextNote)
                found = true
                break
        if !found
            break
        else
            // if we've reached the end of the occurrence
            if i = occurrence.notes.size
                matchedOccurrence = createNewOccurrence(tempNotes)
                matchedOccurrences.Add(matchedOccurrence)

removeDuplicates(matchedOccurrences)
```

Essentially, the algorithm retrieves exact (or chromatically transposed) repetitions of an occurrence by initialising an empty list of notes, `tempNotes`. It then iterates through all of the notes in the MIDI file, attempting to match them to the occurrence. For the very first note of the occurrence, it iterates through all of the notes of the file until it finds one that has an equal duration to the first note. Being part of a prospective match, it adds the note to `tempNotes`.

Then, it iterates through the rest of the MIDI notes following the first matched note, comparing the intervals of each pair of notes with the intervals of the occurrence being matched and verifying that the durations are equal. The reason it compares intervals is that, while pitches may vary in two segments of notes that are transposed from each other, the pitch intervals between each pair of notes will remain the same.

The algorithm uses a helper method, `GetNotesAtTime(time)`, which takes a MIDI time as a parameter and then searches the list of all notes in the MIDI file for any notes that have that precise time. It is used to find all notes that follow the note currently being evaluated by the algorithm. This intuitively allows the algorithm to "move" from one note to another and see if there is a sequence of movements (notes) that follows the occurrence's intervals/durations. It gets *all* notes at a particular time and checks each one, which means that in polyphonic pieces (with multiple notes playing together), the algorithm can still find repeated sequences of notes among the voices.

Each time a note matches the ongoing pitch interval and duration of the occurrence, it is added to the `tempNotes` list. If it reaches a note that isn't a match (it has, compared to the matched occurrence, an unequal duration or different pitch relative to its previous note), it breaks the loop. If not, the loop counter will reach the size of the occurrence's list of notes, which means that every note in the sequence of MIDI notes was a match with the occurrence's notes. With all of those matched notes in the `tempNotes` list, it creates a

new occurrence with those notes and then runs a duplicate check on the occurrence with the other occurrences to make sure it is not a sequence of notes that has been already marked as an occurrence.

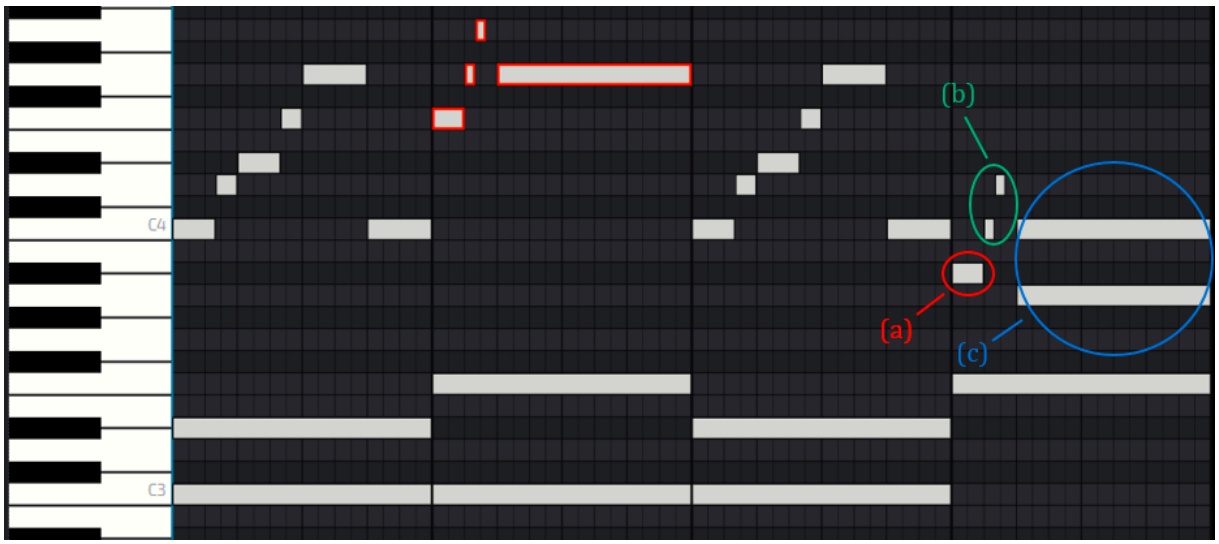


Figure 3.14: An illustration of how the automatic occurrence matching algorithm would approach matching an occurrence.

Figure 3.14 is a labelled screenshot of the annotation tool attempting to match the occurrence whose sequence of notes have a red border. The algorithm iterates through all notes until it finds the first note of the repetition (a), which is where it begins evaluation due to the fact that it has the same duration as the first note of the marked occurrence. It then individually finds the two circled notes (b), and since the pitch intervals between the notes of either repetition are equal and so are the durations, the algorithm continues onto the following notes. It reaches the notes circled in blue (c), and it goes into a loop to evaluate both of them and see if either of them is a matching note. Since the top one matches the occurrence in duration and pitch interval, it finishes its evaluation and collects all of the notes of the new repetition, marking it as a retrieved occurrence.

With the implementation details for ANOMIC's development divulged, the next section goes over the experimental setup and how the user study was implemented.

3.5 User Study

The rest of this chapter offers specifics on the user study designed to evaluate this research.

3.5.1 Overview

The purpose of the user study is to evaluate the performance of ANOMIC and explore the research questions posed in **Section 1.5**. This section will give an overview of the experimental setup.

The user study was designed to be carried out entirely online, and 26 participants were involved of varying demographics and musical/software experience. Participants were contacted through acquaintances, communities of students and musicians, online forums and groups, emails, social media, and instant messaging. All participants performed the user study on their own without any intervention or time limit, with the exception of three who were not adequately familiar with technology and needed some assistance with the configuration.

One can split the experimental setup of the user study into three steps: orientation, annotation, and survey. The next three sections will give some brief information on each section. Finally, the last section will discuss the metrics used for pattern agreement analysis.

3.5.2 Orientation

In the orientation step, users were given a set of step-by-step instructions to read which describe what they are expected to do. These instructions can be seen in **Appendix A**. Included in the instructions is a download to the actual user study[5], an instructional video[6], and the survey[7]. The user study download is a `.zip` file containing the following contents:

- A `.pdf` file with basic instructions accompanied with screenshots illustrating to the user how to get started (such as how to extract the files in the `.zip` file and how to open the annotation tool).
- A `.exe` file to directly run the tool without any installation required.
- Necessary dependency files needed to run ANOMIC (such as `Midi.dll`, the library file used by Midi-Dot-Net).
- Some example `.mid` and `.jams` files showing examples of patterns being annotated in simple melodic passages.
- A folder containing all six pre-made MIDI files within which the participants were expected to annotate patterns using ANOMIC. These MIDI files are excerpts of classical music, the same excerpts used in (Ren, Nieto, et al., 2018), namely:
  - Bach; Cantata BWV 1, Movement 6, Horn (`bach1.mid`)
  - Bach; Cantata BWV 2, Movement 6, Soprano (`bach2.mid`)
  - Beethoven; String Quartet, Op. 18, No. 1, Violin I (`bee1.mid`)
  - Haydn; String Quartet, Op. 74, No. 1, Violin I (`hay1.mid`)
  - Mozart; String Quartet, K. 155, Violin I (`mo155.mid`)

[5]https://tinyurl.com/annotationtooluserstudy
[6]https://tinyurl.com/annotationtoolvideo
[7]https://tinyurl.com/annotationtoolsurvey

– Mozart; String Quartet, K. 458, Violin I (`mo458.mid`)

Double-clicking the executable file loads the application, where users can browse for MIDI files and begin annotation.

3.5.3 Annotation

The next step, annotation, is the process of actually annotating the data. Each MIDI file was annotated by all 26 participants using ANOMIC. There was no time limit, but the user's actions using ANOMIC were logged to the `.jams` files that the participants generated for the user study along with a timestamp for each action. Other than the basic explanation of what a musical pattern is in the user study instructions and some examples, users were not given any other guidelines or strict regulations on what kind of patterns to annotate. Once all MIDI files were annotated, the participants were instructed to send their resulting `.jams` files by email.

The last step of the user study is the survey, where users were asked some questions about their demographic and their experience using the annotation tool. Details will be given in the following section.

3.5.4 Survey

The survey (see **Appendix B**) was split into three sections. The first section's purpose was to glean some data on participant demographics and level of musical experience. Participants were queried on their alias, age, and gender, and then they were prompted to give a general overview of their musical experience. Participants were asked what instrument they play (if any) and how proficient they consider themselves to be. Additionally, they were given a number of statements related to musical experience and asked to mark each statement that applies to them. These statements are to obtain an idea of what type of experience and interaction participants have with music, musicology, musical interfaces, and musical annotation.

The second section prompted users to evaluate ANOMIC based on some standard usability metrics. The metrics being considered are as follows:

- **Learnability**: How quickly can one learn how to use the tool?
- **Performance**: How easy is it to use and perform tasks on the tool?
- **Productivity**: How practical and scalable is the tool?
- **Accessibility**: Is the tool usable by a broad variety of people?
- **User friendliness**: How friendly and robust to errors is the tool?
- **Information scent**: Is the tool's interface complete and clearly presenting all functionality?

- **Flow**: Does the tool flow well from one annotation task to another without workflow interruptions?
- **Customisation**: Can the tool and its interface be customised to the user's liking?
- **User engagement**: How stimulating and fun is the tool for pattern annotation?

The goal of evaluating using these metrics was to gather some qualitative data on the user's experience with multiple different aspects of usability. The last two questions of this section also asked the participant to state three aspects of the tool that they disliked and three aspects that they liked.

The final section dealt with discovering participants' opinions on their experience with ANOMIC and the its functionality. They were first asked to state whether or not they used the automatic occurrence matcher and if they found it helpful. They were then asked if there was any other functionality that they felt was missing from the tool. The next set of questions attempted to find whether or not participants prefer this method of annotation to pen-and-paper annotation. Finally, participants were asked to include any other comments they have about the tool or the study itself.

This user study was carried out in its entirety by 26 participants, and the results will be displayed and discussed in the following chapter.

3.5.5 Pattern Agreement Metrics

One of the aims of the user study was to investigate the levels of agreement exhibited by the annotators, but quantifying agreement in such a task required some consideration. Two different agreement metrics were implemented and explored for the analysis of the annotation data. This section will describe the implementation of these two metrics.

Time Interval Agreement

The preliminary agreement analyses were based on the pattern agreement techniques described in (Ren, Nieto, et al., 2018). In this study, 12 annotators were asked to mark "musical motives" in the same six musical excerpts used for our user study. This was done via pen-and-paper annotation and subsequently digitised and compiled into the HEMAN data set ("GitHub - irisyupingren/HEMANanalysis"). The HEMAN data was then analysed to derive agreement values based on the time interval agreement metric.

This agreement metric operates by matching the time intervals representing the pattern occurrences of the different annotators. A time interval is made up of three components: a start time, end time, and confidence. The start time indicates when the chronologically first note of the occurrence starts sounding, the end time indicates when the chronologically last note of the occurrence stops sounding, and the confidence value is the level of confidence assigned by the annotator (for the annotations being studied in

this project, this is an integer value that is either `1`, `2`, or `3`, with `1` being the lowest and `3` being the highest).

With all occurrences represented as intervals, standard agreement metrics can be measured: precision, recall, and F-score (Goutte & Gaussier, 2005). The formulae for these can be found below:

$$Precision = \frac{\#matchedAnnotations}{\#annotationsOfReferencedAnnotator}$$

$$Recall = \frac{\#matchedAnnotations}{\#annotationsOfCurrentAnnotator}$$

$$Fscore = \frac{Precision \times Recall}{Precision + Recall} \times 2$$

A matched annotation is a pairwise comparison between two annotations from different annotators that resulted in a positive match. A positive match between two annotations A_1, A_2 is defined as follows:

$$\{A_1 = [begin_1, end_1], A_2 = [begin_2, end_2]\} \in \{matchedAnnotations\}$$

if

$$|begin_1 - begin_2| + |end_1 - end_2| \leq Threshold$$

where $Threshold$ is a measure of leniency when comparing annotation intervals. The higher the threshold, the lower the time resolution or the higher the tolerance of the difference between the start and end times of the annotations before two annotations are not considered a match. In the tests, a threshold value of `1` was used, implying one quarter-note's worth of time difference before annotations are not considered matches.

The described time interval metric was utilised to calculate matrices for precision, recall, and F-score, both on the annotations generated from ANOMIC's user study and, for the sake of comparison, on the HEMAN data set annotations. Since the annotation task in this user study (i.e. annotate all repeated occurrences of every discovered pattern) differed from the specific task given to annotators for the HEMAN data set (i.e. annotate musical motives without necessarily marking repeated occurrences), some considerations had to be made to more fairly compare the annotator agreement between the two sets of data.

For the data generated by ANOMIC in our user study, patterns were compared with each other using a grouping technique whereby two patterns (i.e. groups of occurrences) are deemed a match if any single pair of occurrences from the two patterns matches. Conversely, annotations in the HEMAN data set are not marked with repetitions, but rather the annotators were asked to mark just the "most representative" occurrence of

a discovered pattern. Thus, the HEMAN annotations are compared directly without this technique of grouping occurrences together. The results of this agreement analysis between the two data sets can be found in **Section 4.2**.

Note-Level Agreement

Certain downsides of the time interval agreement metric described in the previous section raised questions on its suitability for further analysis of inter-annotator agreement. One of the defining features of ANOMIC is its ability to account for polyphony, and while none of the musical excerpts in the user study featured more than one voice playing at any one time, any future work would be limited to monophonic pieces if time interval agreement would remain the metric of choice. This is because time interval agreement is not able to discriminate between different voices. Annotations simply have a start, an end, and a confidence value. Furthermore, this metric does not support annotations that contain gaps where notes between the start and end time are not supposed to be considered part of the annotation. Essentially, an interval is only able to describe the start and end of an annotation, and all notes within that time interval are assumed to be part of the annotation. Of course, this method could be extended to support more dimensions of interval selection to circumvent these errors, but this was beyond the scope of this thesis.

To aid in future work and provide a more suitable agreement analysis for our user study annotations, different approaches were considered. One such approach would be to take one of the occurrences of an annotator's pattern as a baseline and match that baseline with the occurrences of the other annotator via musical transformation, based on methodology presented in (Melkonian, Ren, Swierstra, & Volk, 2019). Such a technique would generate quantifiable agreement based on how much an occurrence would have to be transformed to match another occurrence perfectly. This promising approach maintains the knowledge of the annotations at a note level, and as such it is robust to the issues mentioned above. However, the transformation technique entails the knowledge of a "base occurrence" for each annotated pattern to properly function, as that is indeed the occurrence that would be used for comparison with other occurrences. Unfortunately, there is no sense of a characteristic or base occurrence in any of the patterns annotated using ANOMIC, and it would be . A pattern is simply a group of related occurrences with no biases on which occurrence is the "reference point" for the pattern. Thus, the transformation technique for pattern agreement was also deemed unsuitable for our specific task.

We therefore present a new metric for pattern agreement. This technique operates at a note level, discarding the notion of time intervals, and it also does not require occurrence transformation or additional knowledge on which occurrence for each pattern can be deemed as a baseline occurrence. The rest of this section will describe in detail how this note-level agreement metric works.

To begin with, the abstraction of patterns being groups of occurrences was not con-

sidered when calculating agreement. We are not looking at the agreement in patterns so much as we are looking at how well the specific occurrences marked by the annotators match with each other, and while this technique could certainly be extended to look into overall pattern agreement rather than just occurrence agreement, this was not considered within the scope of this project.

The first step is, therefore, to define a metric for agreement between two individual occurrences; these will be referred to as O_i and O_j. We perform this task by iterating through the list of all of the notes in the *larger* occurrences (i.e. the occurrence with the greater number of notes) - for the purposes of this example, let us assume the occurrence with more notes is O_j. Each note in O_j would be compared with all of the notes of O_i. Every time an identical note match is found in O_i, this is recorded as a match[8]. If no identically matching note in O_i is found on the occurrence being tested, the note in O_j that is being assessed is deemed to be a mismatch. Thus, the agreement between two specific occurrences can be expressed as a percentage of note matches to total number of notes in the larger occurrence. When comparing equal occurrences, this agreement value would be 100%. Note that since the algorithm always iterates through the occurrence with the *larger* number of notes when comparing two occurrences, calculating agreement for two occurrences is commutative.

Of course, deriving an agreement metric between just two occurrences does not reflect the total agreement between two annotators, as each annotator annotated many different occurrences. Thus, the described note matching process is performed on all possible pairs of occurrences between one annotator and another, generating an agreement matrix with `I` rows representing the occurrences of one annotator, `J` rows representing the occurrences of the other annotator, and the value at `[i, j]` representing the level of agreement between the ith occurrence of one annotator and the jth occurrence of the other. The higher this value, the higher the percentage of notes that were found to have a match.

We now wish to derive a single agreement value out of the entire matrix, which we would use to quantify the level of agreement between different annotators. To achieve this, the most highly agreeing occurrences need to be paired with each other in a mutually exclusive manner. The chosen approach was to use a greedy algorithm that starts by pairing each occurrence of the first annotator to its highest-matching occurrence in the second annotator, generating a list of occurrence pairings, and then it iterates through that list starting from the top. Every time it assigns one occurrence to the other, it marks that row and column in the agreement matrix as being used, meaning that if, further down the list, another occurrence matches the most with an occurrence that was already assigned to a previous occurrence, it does not assign that occurrence and instead saves it in a temporary list. The process repeats itself, with the algorithm now iterating through the temporary list of unassigned occurrences, attempting to assign them again to the next most highly-agreeing occurrence that has not already been used. This continues to loop until all occurrences have been assigned.

With the assignments in place for the occurrences of the two annotators, the algo-

[8]Identical notes are deemed to be notes that have the same time, duration, and pitch values.

rithm at last takes an average of all of the agreement values of each assigned pair of occurrences. Note that since always the first annotator's occurrences are iterated on to assign to the second annotator's occurrences, calculating agreement for two annotators is *not* commutative. As such, if an annotator's set of occurrences A_i all correspond to a unique, exact match in another annotator's set of occurrences A_j, but A_j contains *more* occurrences that are not found in A_i, then running the agreement algorithm $Agr(A_i, A_j)$ would yield a 100% agreement value, whereas running the reverse, $Agr(A_j, A_i)$ will yield an agreement value that is lower than 100%.

To summarise the above, we define pattern agreement as follows:

- A note has a time, duration, pitch, channel, and velocity associated with it. If we consider two notes N_1 and N_2, we say that $N_1 = N_2$ when the values of time, duration, and pitch of both notes are equal.
- An occurrence is made up of multiple notes. Two occurrences O_i and O_j have an agreement value $Agr(O_i, O_j)$ that is calculated via the following formula:

$$Agr(O_i, O_j) = \frac{|\{x \in Notes(O_i), y \in Notes(O_j) \mid x = y\}|}{|Notes(O_i)|}$$

  where O_i is the occurrence with more notes, O_j is the occurrence with fewer notes, $Notes(O_i)$ is the set of notes in the larger occurrence, $Notes(O_j)$ is the set of notes in the smaller occurrence, and the fraction's numerator is the number of items in the set of all note matches between occurrences O_i and O_j.
- An annotator is associated with a set of occurrences for a given music file. There can be any number of occurrences, and using the above occurrence agreement metric we can generate an agreement matrix where each row is the first annotator's set of occurrences, each column is the second annotator's set of occurrences, and the corresponding matrix value is the calculated level of agreement.

  Deriving the singular agreement value from this matrix involves iterating through the first annotator's occurrences (the rows), finding the highest matches with the second annotator's occurrences (the columns), and finally assigning each occurrence uniquely in a manner that maximises occurrence agreement. The pseudocode of this algorithm can be found below.

```
// assign a maximally agreeing occurrence to each of the first annotator's occurrences
List<Assignment> remainingList = GetMaxOfEachRow(agreementMatrix)
List<Assignment> assignments = new List<Assignment>

do
    sortedList = Sort(remainingList)
    usedColumns = new List<MatrixColumn>
    remainingList = new List<Assignment>

    // iterating through the sorted occurrence assignments
    for each currentAssignment in sortedList
        if usedColumns.Contains(currentAssignment.column)
```

```
            remainingList.Add(currentAssignment)
        else
            usedColumns.Add(currentAssignment.column)
            assignments.Add(currentAssignment)

            // mark all values in the used column as 0
            for i = 1; i < agreementMatrix.RowCount; i++
                agreementMatrix[i][currentAssignment.column] = 0;

    List<Assignment> tempList = remainingList.Clone()
    remainingList.Clear()

    // update the remaining list with new assignments
    // now it will not assign already-used columns as the values are all marked as 0
    for each assignment in tempList
        remainingList.Add(GetMaxOfRow(agreementMatrix, assignment.row))
while remainingList.Count > 0
```

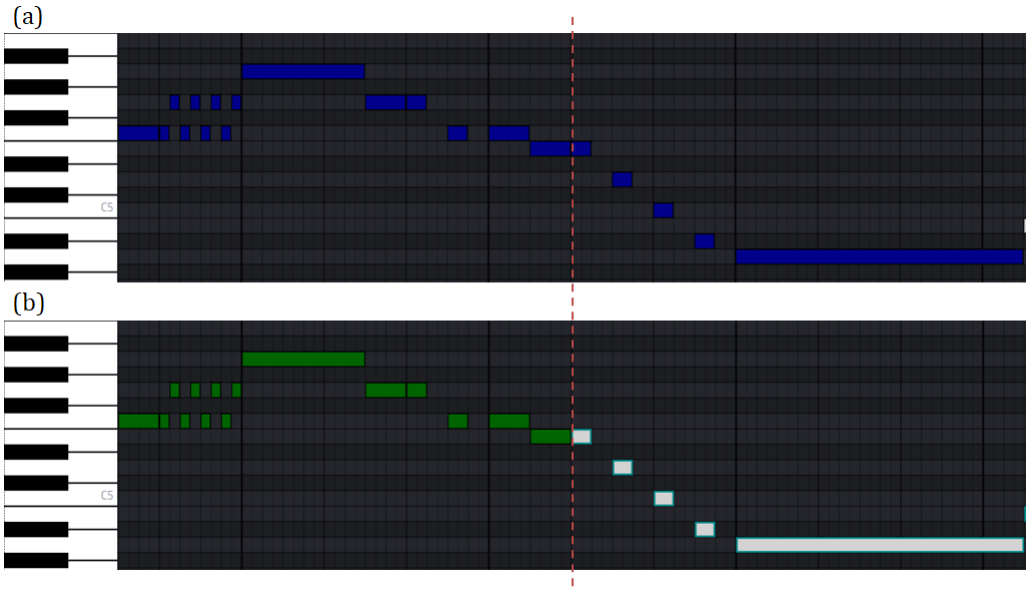


Figure 3.15: A comparison between two similar occurrences (a) and (b) annotated by different annotators.

Finally, for clarity, consider **Figure 3.15** as an example of the note-level comparison performed by the described agreement metric between two occurrences. This figure compares a sequence of notes extracted from one of the excerpts (`bee1.mid`) as annotated by two different annotators. The first annotator (a) marked the entire sequence of 20 notes as a single occurrence. The second annotator (b) marked up until the dotted line – the first 15 notes – as a single occurrence (deeming the rest of the notes as being part of another occurrence). When the algorithm reaches the point of needing to compare these two annotations from different annotators together, it first considers which occurrence has the larger number of notes. With 5 more notes than its counterpart, (a)'s occurrence is deemed to be the larger one. It iterates through all 20 notes of (a)'s occurrence, noting each time a note in (a)'s occurrence can be exactly matched to a note in (b)'s occurrence.

It finds 15 matches. As per the above formula, it divides the 15 note matches by the 20 notes in the larger occurrence, yielding an agreement score of 75%.

This process is repeated for every possible occurrence pairing between annotators (a) and (b), and the highest-matching occurrence pairs are assigned. Finally, the agreement values of all occurrence pairs (using the note-level agreement technique described) are averaged to determine a final agreement score between the two annotators.

The next chapter goes into the results of the user study and ANOMIC's evaluation.

Chapter 4

Results

This chapter displays statistics and performs studies on the results of the user study whose design and implementation was described in **Section 3.5**. Both the survey and annotation results will be considered in this chapter, and the results will be discussed and compared to draw conclusions on the data and the quality of the annotation tool.

We will first output and discuss some quantitative results of the annotations done by the participants. Afterwards, there will be a section specifically for discussion on the agreement analysis. Finally, this will be followed by discussion on the quantitative and qualitative results of the user study's survey.

4.1 Statistical Overview

This section will provide a general statistical overview and discussion of the annotation results. In all displays of the results, the participants have been represented as numbers from 1 to 26 to preserve anonymity.

Table 4.1 shows results of the user study annotations for each participant. Column 1 (Part) is the index of the participant; column 2 (Pat) is the number of patterns they annotated for all six MIDI files; column 3 (Occ) is the number of occurrences they annotated for all of their patterns; finally, column 4 (Note/Occ) shows the average number of notes the participant selected per annotated occurrence, and was calculated with the following formula:

$$Avg_{notes} = \frac{\sum_{i=1}^{Num_{Occ}} Notes_{Occ_i}}{Num_{Occ}}$$

where Avg_{notes} is the average number of notes per occurrence annotated by a participant (Note/Occ), Num_{Occ} is the total number of annotated occurrences, and $Notes_{Occ_i}$ are the number of notes selected for the ith occurrence. The final row displays the averages and totals of each column.

Part	Pat	Occ	Note/Occ
1	30	98	7.37
2	26	78	7.22
3	26	79	8.59
4	18	55	6.56
5	29	113	6.99
6	32	95	8.63
7	40	79	7.05
8	25	73	8.3
9	18	45	9.24
10	23	62	8.66
11	26	100	6.93
12	17	77	10.91
13	26	80	7.58
14	36	110	5.63
15	35	114	7.95
16	75	306	3.58
17	26	73	8.27
18	38	113	6.03
19	12	34	4.06
20	28	93	7.4
21	29	91	8.34
22	27	86	9.09
23	21	62	11.26
24	9	45	9.58
25	28	77	6.94
26	88	525	2.28
Avg	**30.31**	**106.27**	**7.48**
Tot	**788**	**2763**	**194.44**

Table 4.1: A table of participants' overall statistics: number of patterns annotated, number of occurrences annotated, and average number of notes per occurrence.

The above table gives a good general idea of the quantity and quality of the annotations. Most participants annotated within the range of under 100 or slightly over 100 occurrences, with the exceptions of participants 16 and 26 who annotated 306 and 525 patterns respectively.

Similarly, **Table 4.2** showcases statistics of the user study's annotations on each file. Column 1 (File) is the name of the MIDI file; column 2 (Dur) is the duration of the MIDI file in the format `MM:SS` where `M` is a minute digit and `S` is a second digit; column 3 (Pat) is the total number of patterns annotated by all 26 participants on that file; column 4 (Occ) is the total number of occurrences annotated on that file; finally, column 5 (Note/Occ)

shows the average number of notes in an occurrence annotated for that file.

File	Dur	Pat	Occ	Note/Occ
bach1	01:04	100	254	10.23
bach2	00:30	78	229	4.86
bee1	01:30	138	698	6.42
hay1	00:58	137	452	5.22
mo155	00:55	151	399	6.69
mo458	01:21	184	731	5.61

Table 4.2: A table of the files' overall statistics: name of file, duration, total number of annotated patterns, total number of annotated occurrences, and average number of notes per occurrence.

We note that most files have an average notes per occurrence value roughly within the range of 5 to 6 notes, but that `bach1.mid` possessed a much higher value. It was postulated that this was the case primarily because of how the particular excerpt was structured musically. There is an exact repetition of the first 46 notes directly afterwards. Many annotators opted to count the entire 46-note segment of the excerpt as a pattern due to this phenomenon, which greatly skewed the Note/Occ value for that particular file.

Table 4.3 provides the time that each participant took to annotate all six files. This time was calculated by considering time intervals between each user action logged in the .jams files and adding each time interval to the total time spent.

As the participants were not supervised or informed that they were being timed, an inactivity threshold of 120 seconds was determined, which meant that every time there was more than two minutes in the time interval between any two user actions, it was not added to the overall time spent annotating. This was to avoid any undesirable biases in the results caused by people stopping to take breaks or getting distracted. For posterity and transparency, the total "inactivity time" was also included in the table.

Column 1 (Part) of the table is the index of the participant; column 2 (Time) is the total time spent annotating; column 3 (Inac) is the inactivity time that was not counted with the annotating time. The bottom row shows the average of each of the two columns. Note that the time in Column 2 and 3 is formatted as `HhMMmSS.IIIs`, where `H` is an hour digit, `M` is a minute digit, `S` is a second digit, and `I` is a millisecond digit.

Part	Time	Inac
1	1h01m08.020s	0h05m20.590s
2	0h35m18.990s	0h06m00.510s
3	0h38m39.150s	0h07m50.780s
4	0h29m58.450s	0h00m00.000s
5	0h43m05.180s	0h02m04.010s
6	0h36m18.850s	0h05m19.630s

7	0h30m40.240s	0h06m31.730s
8	1h13m47.820s	0h27m35.170s
9	0h27m04.150s	1h13m10.190s
10	0h46m53.020s	0h00m00.000s
11	0h41m06.990s	0h04m52.720s
12	0h33m47.040s	0h24m09.510s
13	0h37m27.270s	0h00m00.000s
14	0h33m23.660s	0h05m05.230s
15	1h05m36.960s	0h03m19.950s
16	0h36m58.770s	0h00m00.000s
17	0h26m23.140s	0h03m25.170s
18	1h24m33.130s	0h02m37.320s
19	0h07m27.450s	0h00m00.000s
20	0h16m30.080s	0h03m19.660s
21	0h34m20.030s	0h50m41.830s
22	0h36m58.850s	0h00m00.000s
23	0h19m22.240s	0h24m20.940s
24	0h27m30.230s	1h59m34.320s
25	0h39m19.480s	0h08m20.550s
26	0h45m05.250s	1h29m31.330s
Avg	**0h38m47.863s**	**0h18m11.967**

Table 4.3: A table of participants' time taken and inactivity time on all of the six annotated files.

Similarly, **Table 4.4** shows the average amount of time a participant would take for each file. Column 1 (File) is the name of the MIDI file; column 2 (Time) is the average time a participant took on that file.

File	Time
bach1	0h07m19.940s
bach2	0h04m34.193s
bee1	0h08m43.964s
hay1	0h05m25.455s
mo155	0h05m40.799s
mo458	0h07m03.513s

Table 4.4: A table of average time taken by a participant per file.

The most notable outlier from this data is participant 19, who only spent 7 minutes annotating compared to the average annotation time of nearly 40 minutes. Due to the nature of the user study being performed online and with no control over the annotation

session, it is to be expected that certain annotators might not exhibit close to the required effort or time.

Using the above data, we illustrate a graph comparing time taken to number of annotated occurrences. **Figure 4.1** is a scatter chart that plots every single participant's time taken in seconds on the y-axis and the number of annotated occurrences on the x-axis. A trend line that indicates the overall course of the set of points on the graph has also been illustrated. The trend suggests that higher quantities of occurrences may indicate that a longer time was taken to annotate them all.

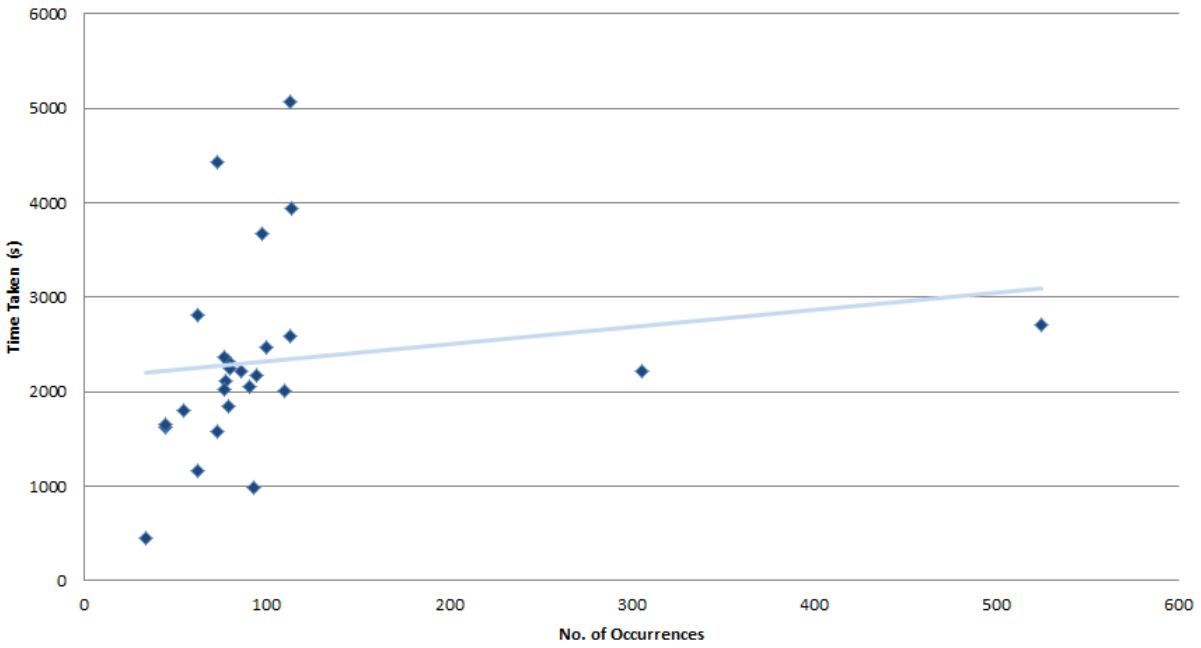


Figure 4.1: A scatter chart that plots time taken on the y-axis vs number of annotated occurrences on the x-axis.

The graph illustrates the aforementioned two outliers in number of occurrences: participant 16 and participant 26. Both of these annotated far more occurrences than everyone else despite not necessarily having put much more time into their annotation than others, and they also have far fewer average notes per occurrence (3.58 and 2.28 respectively). This radical change from the general trend of the other data points suggests that these two annotators might not have the same understanding of what constitutes a pattern as others did. A clearer set of guidelines in the instructions and more close supervision may have been beneficial in this case.

A highly insightful analysis would be to compare the time taken to annotate using ANOMIC with the time taken to annotate using the traditional pen-and-paper approach (with time taken to digitise everything included). We could then directly compare how quickly users would take to annotate using the different methods. Unfortunately, due to time constraints, lack of available data, and concerns about availability of interested participants, no study was performed in this research project which involved annotating using the traditional approach, which makes comparison difficult in this regard.

The next section deals with studies into inter-annotator agreement and comparison with the agreement data found in the HEMAN data set ("GitHub - irisyupingren/HEMANanalysis").

4.2 Agreement Results

This section shows and discusses the results of analysis into pattern agreement using the metrics described in **Section 3.5.5**.

The first metric that was explored was the time interval metric. Results found using this metric for both the HEMAN data set and the data used in our user study can be found in **Appendix D**. In this appendix, precision, recall, and F-score matrices are displayed separately for each of the six files and for both of the data sets. The rows and columns in the matrices correspond to the different annotators, and for visualisation purposes, a colour scale whose brightness scales linearly with the agreement value was used.

With there being 12 participants in the HEMAN data set study, results for agreement metrics performed on that data are in the form of 12x12 matrices. Similarly, with our user study reaching 26 participants, the matrices are thus 26x26. If `i` is the row index and `j` is the column index, then the value at `[i, j]` in any matrix represents the precision, recall, or F-score value that the ith annotator has with the jth annotator. As discussed in **Section 3.5.5**, this relation is not necessarily commutative. Naturally, every participant agrees with themselves with a 100% level of agreement, so the diagonal values of each matrix will be 100%.

Table 4.5 extracts more readable and concise data from the matrices. For both the ANOMIC and HEMAN data, the F-score matrix of each excerpt was averaged to obtain a total F-score value for all annotators. Column 1 (File) is the name of the MIDI file; column 2 (Agr) is the average F-score of all annotators on that file from the ANOMIC data; column 3 (File) is the name of the MIDI file; finally, column 4 (Agr) is the average F-score of all annotators on that file from the HEMAN data.

ANOMIC		HEMAN	
File	**Agr**	**File**	**Agr**
bach1	0.28	bach1	0.25
bach2	0.38	bach2	0.31
bee1	0.48	bee1	0.32
hay1	0.36	hay1	0.26
mo155	0.4	mo155	0.29
mo458	0.45	mo458	0.24
Avg	0.39	**Avg**	0.28
Dev	0.07	**Dev**	0.03

Table 4.5: A table of the average of all F-score values per file for the annotation data of both data sets.

It can be seen that the agreement in our annotation data using ANOMIC significantly improves upon the agreement in the HEMAN data set. While agreement in both cases mostly followed similar trends (both sets of annotators struggled with `bach1.mid` but achieved high levels of agreement on `bee1.mid`), agreement levels across the board are higher. While this reflects positively on the annotations made with ANOMIC, there is some notable discussion to be made about the comparison method used.

As mentioned in **Section 3.5.5**, the results of the ANOMIC data and the HEMAN data cannot be directly compared, due to the difference in annotation tasks. While ANOMIC's annotations are meant to include all repetitions of each discovered pattern, the annotators behind the HEMAN data set were simply told to find representative musical motives without marking repetitions. A grouping technique was used where each discovered pattern in the ANOMIC data was treated as a group of occurrences and as long as one occurrence in this group matched with another occurrence in another pattern's groups of occurrences, then it was said that both patterns are in agreement. This technique is not applicable to HEMAN's data.

The result is that ANOMIC's annotations have more chances to agree with each other than HEMAN's annotations. An annotator for HEMAN's data set can annotate one repeated occurrence of a pattern while another annotator marks a different repeated occurrence of the very same pattern, leading to disagreement, whereas in ANOMIC's case, both annotators are expected to mark all repetitions of a pattern, meaning that if at least one pair of marked repetitions between the two annotators' groups of occurrences is found to be a match, then the patterns are altogether deemed to be a match. Essentially, this biases the results to favour ANOMIC's structure of data.

Despite this bias, in essence we can see that, using the time interval agreement metric, annotators did tend to agree with each other in ANOMIC's data on which patterns to annotate. The similar trends between the results of the two data sets also suggests that annotations done using ANOMIC are certainly comparable and of similar quality to annotations done using sheet music.

The next step is to look into results found using the note-level agreement metric described in **Section 3.5.5**. The agreement matrices for these results can be found in **Appendix E**. **Figures E.1, E.2, E.3, E.4, E.5, and E.6** illustrate the matrices for annotation done on `bach1.mid`, `bach2.mid`, `bee1.mid`, `hay1.mid`, `mo155.mid`, and `mo458.mid` respectively. Once again, the rows and columns are used to denote annotators, and a colour scale was used to aid visualisation.

A significant finding here would be that once again, participants 16, 19, and 26 are shown to be outliers. 16 and 26 were the two participants with the most annotated occurrences and the fewest average notes per occurrence by a significant margin, and this is reflected in the matrices. Their rows consistently contain low agreement scores, because when matching their occurrences to other annotators' occurrences in a mutually exclusive manner, the algorithm quickly ran out of occurrences to match to from the other annotator. Thus, participants 16 or 19 attained low scores due to so many of their occurrences not having any match at all. Conversely, participant 19 was the participant with the

lowest time spent by far (and a subsequently low number of annotated occurrences). The few occurrences participant 19 annotated fit very often with the other annotators' occurrences, so row 19 in each matrix had fairly high agreement values, but conversely, when comparing other participants to participant 19, there were too few occurrences annotated by that participant for there to be a high agreement.

These discrepancies in agreement are shown more clearly in **Figure 4.2**, where the average of all values in the annotator's corresponding row and column were taken and averaged to yield a more generalised agreement score for each annotator. These agreement averages are illustrated in the chart shown below, where we can clearly see the poor annotator agreement exhibited by these three participants.

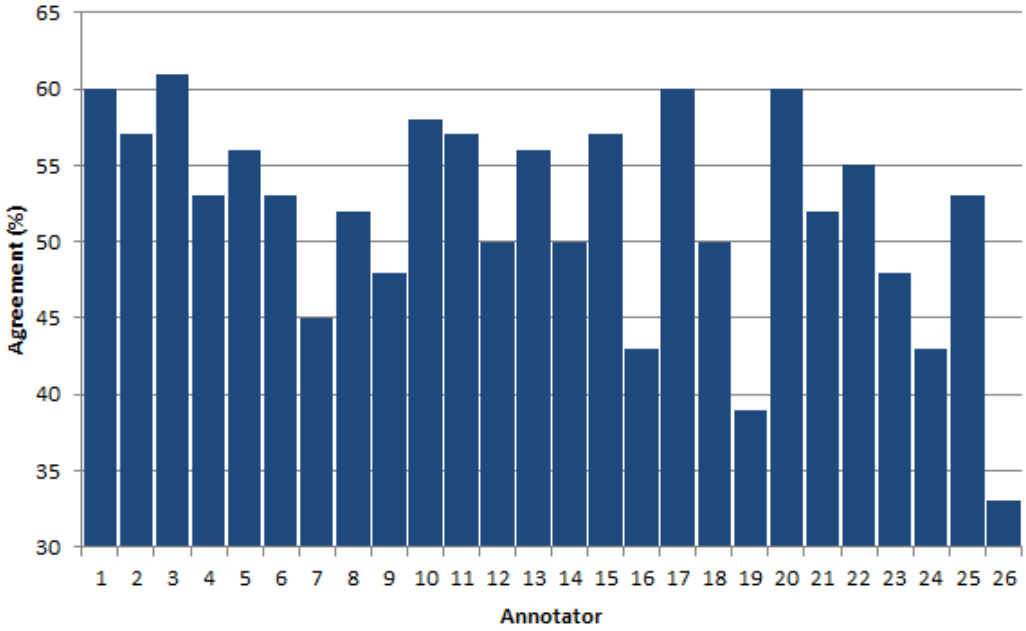


Figure 4.2: A column chart with average note-level agreement on the y-axis and annotator index on the x-axis.

The next study into inter-annotator agreement shows how note-level agreement varied across the different musical excerpts. **Table 4.6** is a table showing all of the six MIDI files used in the user study along with the average agreement that the annotators attained on each one. This average agreement was calculated by simply taking the average of all cells in the agreement matrix for each file. Column 1 (Name) lists the name of the MIDI file; column 2 (Agr) lists the average agreement value for that file as a percentage.

As is apparent from the table, `bach1.mid` possesses the lowest agreement score. This reflects the low agreement score attained in the time interval agreement results of **Table 4.5**. This could partially be due to the fact that, with it being likely the first file that all of the annotators would be annotating on, they would have not had time to adjust to using the tool and may be more prone to making mistakes or annotating more carelessly. That said, as stated in (Ren, Nieto, et al., 2018), `bach1.mid` possessed structural ambiguity and very little rhythmic variation or clear grouping cues, making it arguably more difficult

than the rest of the files for annotation. Another potential factor for a lower level of agreement is that, as per **Table 4.2**, `bach1.mid` had a much higher number of notes per occurrence than all other files.

Name	**Agr**
bach1	41%
bach2	51%
bee1	67%
hay1	48%
mo155	53%
mo458	51%

Table 4.6: A table showing the average agreement percentages among all annotators on each MIDI file.

Conversely, `bee1.mid`, being one of the shortest files, had a significantly higher average agreement than all other files. Unlike `bach1.mid`, this excerpt is relatively simple and straightforward musically, and due to it being so short, there was less musical content for the participants to digest. Furthermore, it contained very clear and obvious motifs that could be assigned as patterns. It was postulated that these reasons made it easier for annotators to create a more complete and unanimous annotation of the excerpt without getting overwhelmed by length or missing out on obvious repetitions.

Looking now to the agreement differences at an annotator level, one important comparison to make is the difference in agreement between musicians and non-musicians, the hypothesis being that people with a richer musical background will agree with each other more than those with less of a musical background. To research this hypothesis, the database of annotations was segregated into two: those who scored a musical background rating[1] of 4 or below and those who scored a musical background rating of 5 or above (see **Table 4.10**. This split the data set into 12 "non-musicians" and 14 "musicians" respectively. The agreement matrices were calculated from scratch for these subsets of the data set and the average agreement per file was found for each subset. This data can be found in **Table 4.7**. Column 1 (Name) lists the name of the MIDI file; column 2 (AgrN) lists the average agreement per file for the"non-musicians"; column 3 (AgrM) lists the average agreement per file for the "musicians".

As one can see, with the curious exception of `bach2.mid`, those with a richer musical background demonstrated significantly higher average agreement per file. The last row of **Table 4.7** also shows the total average agreement for "non-musicians" versus "musicians", where it is apparent that those with more musical experience agreed with each other by a factor of 16% more than those with a generally lower level of musical experience. As covered in **Section 1.3.2**, pattern agreement among annotators is generally seen as

[1]The musical background rating is calculated by counting the number of statements the users checked in question 4 of the survey and adding their instrument proficiency value from question 6. More detail can be found in **Section 4.3.1**.

something to strive for, so these results indicate that people with more musical experience are able to use the annotation tool to greater effect.

Name	**AgrN**	**AgrM**
bach1	36%	49%
bach2	57%	50%
bee1	66%	70%
hay1	47%	53%
mo155	47%	62%
mo458	46%	61%
Avg	**50**	**58**

Table 4.7: A table showing the average agreement percentages per file for non-musicians vs musicians, with average values displayed on the final row.

Finally, one of the more prominent features of ANOMIC (and indeed one of the advantages that a digital annotation tool would have over sheet music) is the automation of the discovery of repeated occurrences. The automatic occurrence matching feature[2] is able to comb through the entire MIDI file and find transposition-invariant, polyphony-robust exact repetitions of a given occurrence. This was used to varying extents by the participants, and **Table 4.8** shows how many times each participant used the automatic occurrence finder across all six files, with column 1 (Part) being the index of the participant and column 2 (Uses) specifying the number of uses that participant had of the occurrence finder feature.

Part	**Uses**
1	25
2	12
3	50
4	32
5	36
6	23
7	48
8	29
9	54
10	15
11	28
12	12
13	25
14	40
15	30
16	122

[2]Please refer to **Section 3.4.3** for more details on this feature.

17	40
18	60
19	25
20	9
21	13
22	22
23	9
24	13
25	25
26	250

Table 4.8: A table indicating how many times each participant used the automatic occurrence finder feature.

How the agreement varies with how frequently the automatic occurrence finder was used was also explored. The 13 participants that used the feature the least and the 13 that used the feature the most were split up and agreement averages per file were calculated for each group separately. The results of this study can be seen in **Table 4.9**. Column 1 (File) is the name of the excerpt; column 2 (AgrM) is the note-level agreement value computed for manual users (i.e. the top 13 that used the automatic occurrence finder the fewest times); column 3 (AgrA) are the computed agreement values for the rest of the 13 participants that used the automatic occurrence finder the most.

File	**AgrM**	**AgrA**
bach1	38%	23%
bach2	46%	37%
bee1	52%	46%
hay1	34%	46%
mo155	42%	44%
mo458	48%	49%
Avg	43%	41%

Table 4.9: A table showing how much users who prefer manual discovery of patterns agree with users that erred towards automatic discovery.

One can see that those that used the automatic occurrence finder the fewest tended to agree slightly more than those who annotated using the automatic occurrence finder tool the most. One reason for this may be that those that preferred the automatic occurrence finder are likely annotators that do not trust their own instincts to find repeated occurrences themselves and may be less well-versed or experienced with this annotation process.

However, it is to be noted that participants 16 and 26 skew the data significantly,

having used the automatic occurrence finder tool far more often than everyone else. Coupled with the fact that they annotated many more occurrences with much fewer notes than all other annotators, it is clear that they may have misunderstood the annotation task that is being asked of them. Instead of finding musically sensible segments of the excerpts, they opted to annotate very short musical phrases of 2-4 notes each and use the automatic occurrence finder to find exact transposition-invariant repetitions of their small and musically insignificant sets of notes. Once again, closer supervision and clearer instructions could have avoided this situation and possibly resulted in less skewed data.

In fact, with both outliers removed and the agreement values computed again, the average note-level agreement for users who prefer the automatic occurrence finder bumps up to **46%**, which is a marked improvement from its previous value of 41% with outliers included and indicates moderately improved agreement over those that used the feature less.

The next sections will look into survey results.

4.3 Survey Results

4.3.1 Quantitative Results

This section will go over the quantitative results obtained from the survey that was sent out to all participants after they completed the annotation section of the user study. The survey questions can be found in **Appendix B** and will be referred to regularly over the course of this section.

After the consent question, the first six questions deal with user demographics. Participants were queried on their alias (purely for the purposes of matching the annotation data to the user survey data), age, gender, and some basic questions on musical background. Most of these demographic results are summarised in **Table 4.10**. Column 1 (Part) is the index of the participant; column 2 (Age) is the age survey choice chosen by the participant; column 3 (Gender) is the gender choice chosen by the participant; finally, column 4 (Mus) is a rating of the participants' musical background. The musical background rating was determined simply by counting the number of items chosen in question 4 of the survey and adding that result with the self-assessed instrument proficiency value chosen by the participant in Question 6 (if applicable). For example, if a survey participant checked "I enjoy listening to music." and "I can read sheet music." and stated that they play piano with a proficiency of 3, then the resulting musical background rating would be 5. While being a fairly crude method of quantifying a participant's musical background, it serves the purposes of this research.

Part	Age	Gender	Mus
1	41 - 60	Male	6
2	17 - 21	Male	8

3	41 - 60	Male	3
4	41 - 60	Male	4
5	22 - 28	Male	5
6	22 - 28	Male	10
7	29 - 40	Prefer not to say	1
8	61 or above	Male	10
9	17 - 21	Female	7
10	41 - 60	Female	6
11	17 - 21	Female	2
12	22 - 28	Male	4
13	17 - 21	Male	1
14	29 - 40	Male	6
15	22 - 28	Female	2
16	16 or under	Male	4
17	17 - 21	Male	5
18	41 - 60	Female	1
19	41 - 60	Male	4
20	29 - 40	Other	5
21	17 - 21	Male	1
22	22 - 28	Female	5
23	22 - 28	Female	10
24	22 - 28	Male	6
25	41 - 60	Female	5
26	17 - 21	Male	2

Table 4.10: A table showing participant demographics, namely the age, gender, and musical background rating.

The next section of the survey (i.e. questions 7–15) dealt with usability metrics. Users were asked to give a rating from 1 to 5 for each usability metric from a standard list of metrics detailed in **Section 3.5.4**. Lower values are considered unfavourable, as they imply that the participants thought the tool performed worse under the given criteria.

Figure 4.3 shows nine bar charts, each one corresponding to one of the usability metrics being tested in the survey. The charts shown are participants' scores on learnability (a), performance (b), productivity (c), accessibility (d), user friendliness (e), information scent (f), flow (g), customisation (h), and user engagement (i). In each chart, the y-axis contains these numerical ratings, while the x-axis is a count of how many participants chose a given rating.

Table 4.11 also shows different statistical measures of this data. Column 1 (Metric) is the name of the usability metric; column 2 (Avg) is the average of all participant scores on that metric; column 3 (Dev) is the standard deviation of the participant scores on that metric.

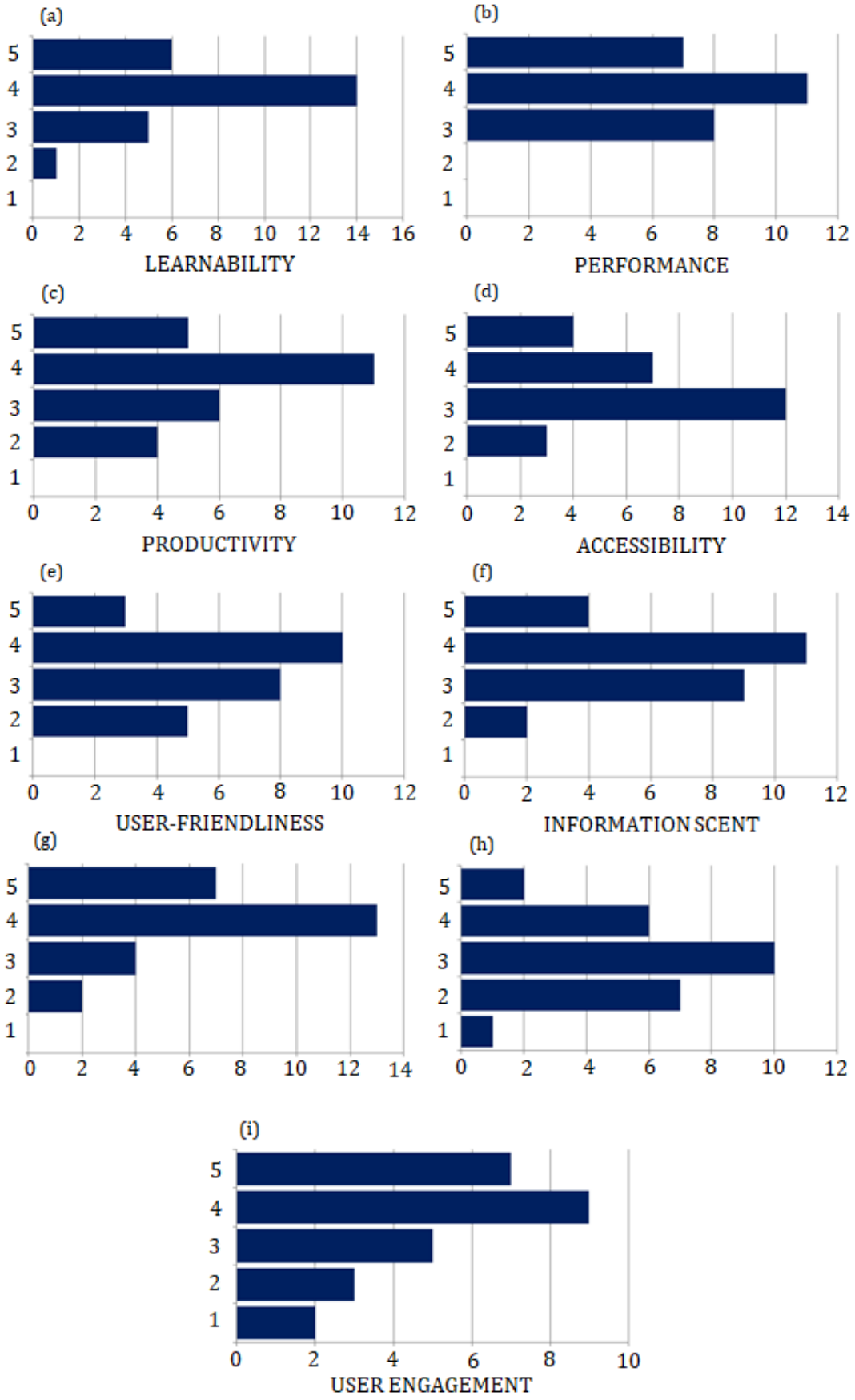


Figure 4.3: A set of nine bar charts with metric ratings on the y-axes and the count of participants who chose those ratings on the x-axes.

Metric	Avg	Dev
Learnability	3.96	0.77
Performance	3.96	0.77
Productivity	3.65	0.98
Accessibility	3.46	0.90
User friendliness	3.42	0.95
Information scent	3.65	0.85
Flow	3.96	0.87
Customisation	3.04	1.00
User engagement	3.62	1.24

Table 4.11: A table showing the average and standard deviation of participant scores on usability metrics.

The last part of the survey that will be looked into in this section is question 20, where users are asked to provide a rating from 1 to 5 that describes how much they would prefer using the traditional pen-and-paper annotation method or the annotation tool, with a rating of 1 implying they would strongly prefer the traditional approach and a rating of 5 indicating that they would strongly prefer the digital approach with the annotation tool. Answers from this question can be seen in **Figure 4.4**.

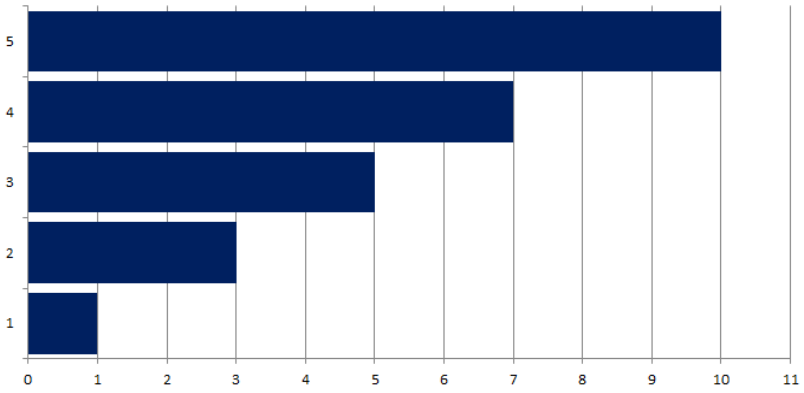


Figure 4.4: A bar chart with preference rating on the y-axis and count of participants who chose those ratings on the x-axis.

The above findings will be discussed in the next section.

4.3.2 Qualitative Survey Results

Some of the more insightful answers from the survey results were in response to question 16 and 17, where users were asked to state three aspects of the tool they thought could be improved and three aspects that they liked the most. In the former question, common issues that were brought up include:

- The tool lacks in user friendliness. There was no help menu, tool tips, hint balloons/arrows, etc.
- The interface could be more clear and usable; there is no visual tell for differing confidence levels of occurrences, scrolling is difficult without the middle mouse button due to small scroll bars, certain buttons are small and difficult to click, occurrence and pattern icons in the left panel could be made larger when they are active, and certain graphics could be made more visually appealing.
- The tool lacks some minor features that make the experience smoother, including:
  - The viewing box of the interface was not scrolling as the music plays.
  - Occurrences that have already been finalised cannot be edited and would have to be deleted and remade.
  - There is no undo/redo feature.
  - There is no ability to play a single pattern on its own.
  - There is a lack of MIDI pedal support.
  - Pattern colours are fixed and not customisable.
  - The availability of hotkeys was limited.
  - The tool does not have any unique project file formats; loading an annotated MIDI file involves first loading the MIDI file and then loading its respective JAMS file, instead of loading a single project file.
- The automatic occurrence finder could be expanded upon more to include the ability to find inexact matches or rhythmic patterns.
- The tool was prone to crashes, bugs, and performance issues.

The most common criticisms brought up by the participants are the user friendliness being somewhat lacking, there being no undo and redo functionality, there being no option to customise pattern colours or titles, and there being no option to allow the music to automatically scroll as it is being played. These issues are in accordance with the fact that user friendliness and customisation usability metrics have the two least metric scores as per **Table 4.11**.

Furthermore, one of the more notable issues brought up by the participants was the fragile nature of the tool; certain actions would cause the tool to crash. At the time of writing this paper, these bugs have been patched and were due to improper/incomplete testing. In addition to the above criticisms, some issues also reported that the tool lacked in certain features that were actually present in the tool, such as the ability to zoom, the option to place the scroll bar at the position of a clicked occurrence, and the ability to normalise note volume across the piece. These are all features that are already available in the tool, but the presence of these responses suggests that these features are not clear or accessible enough for everyone to be aware of their existence.

Conversely, question 17 asked participants to state the features or aspects of the tool that they liked the most. A summary of the responses to this question can be found below:

- The ease of use after learning how the tool works was praised by many.
- The smooth and usable audio playback feature was said to be very helpful during annotation.
- The pattern colour-coding was generally very well received.
- Despite some downsides, the interface was deemed pleasant to navigate, clean, and visually appealing.
- Some users stated that the tool provided a more compelling and engaging method to annotate music than the traditional pen-and-paper approach.
- The automatic occurrence finder was almost universally praised by the participants.

The positive aspects of the tool that were brought up reinforce the main draws for developing a digital annotation tool discussed in **Section 1.4**, especially the fact that many enjoyed the audio playback functionality and the automatic occurrence finder, both features that would not be available in a purely pen-and-paper approach. The fact that many enjoyed the interface for its simplicity, aesthetic appeal, and responsiveness suggests that, despite criticisms brought up towards the interface in the previous question, using the current interface as a baseline to start from would be viable.

Question 18 queried users on the use of the automatic occurrence finder. This feature would allow the user to right-click on any occurrence they have found and automatically list all other exact matches in the piece (in a transposition-invariant manner). As the responses to question 17 suggest, the vast majority of the participants (22 out of 26) readily used the automatic occurrence finder, with one participant stating that he occasionally used it but found it lacking for rhythms and variations, and three remaining participants stating that they did not use the automatic occurrence finder at all. Of those three participants, one of them (a seasoned musician with a musical background score of 10) stated that she relied on her eyes and ears to find the patterns, and another one stated that she did not use the automatic finder as it "felt like cheating".

Among the participants that did use the automatic occurrence finder, one participant with the lowest possible musical background score demonstrated a potential over-reliance on the finder, stating that they would have not been able to find pattern occurrences at all without using this feature. There was no definitive consensus on whether or not the automatic occurrence finder was adept at finding many repetitions; some participants stated that it found many exact occurrences, while others reported that there were not many scenarios where the automatic occurrence finder found results. This controversy is likely due to the fact that the quantity of occurrence matches it finds depends very heavily on the initial occurrence that the user finds matches on. Shorter patterns, for instance, will more likely have many exact occurrences within the piece than longer patterns.

Question 19 asked users to think of any features that they felt were missing from the tool. Some promising ideas that were not already mentioned included showing a "pattern match percentage" near each occurrence that quantifies how much it matches with the first-selected occurrence, the ability to name occurrences or patterns, and the ability to see sheet music alongside the piano roll interface. The latter suggestion was indeed already planned as a possible feature for the tool in early stages of development, but due to time constraints it never saw fruition.

Finally, question 21 asked participants to explain their rating given on question 20 (which asked participants to state whether or not they would prefer using the annotation tool over the pen-and-paper approach). 5 out of the 26 participants remarked that the piano roll interface provided a method for users that cannot read sheet music to still use the tool. 15 out of the 26 participants outright stated that the annotation tool allowed for annotation that was quicker, easier, and/or more precise. 4 out of the 26 participants asserted that they would prefer the pen-and-paper approach; of those 4, one of them indicated that he was accustomed to piano roll interfaces and all four of them indicated that they were able to read sheet music. 7 other participants stated that they were accustomed to piano roll interfaces and all of them had an either neutral (3) or positive (4+) preference towards annotating using the annotation tool over the pen-and-paper approach. The benefits mentioned in the responses once again agree with the arguments presented in **Section 1.4** against the traditional annotation method, especially that the digital approach is more scalable and efficient.

With the results displayed and discussed, the final chapter that concludes the thesis and outlines future work and improvements will follow.

Chapter 5

Conclusion

This chapter will summarise the contribution of this thesis and the findings of the user study that was carried out, along with outlining any potential future work that can be done with this project as a basis. First, ideas for the aforementioned future work will be presented. Subsequent to that, the thesis will be closed off with a conclusion summarising the entire project.

5.1 Future Work

While much consideration and work was put into the development and evaluation of the annotation tool ANOMIC, there is always room to reflect on how to improve and build upon the implementation and research carried out.

A lot of implementation work can be performed to improve ANOMIC itself. Many useful features suggested by participants in the user study's survey are worthy candidates to be implemented as official features, such as auto-scrolling when the music is playing, editing existing occurrences, providing undo/redo functionality, adding a "preview occurrence" playback button near each occurrence, supporting unique project file formats that merge MIDI and JAMS files, adding more customisation to the interface, supporting undo/redo options, or of course providing support for more formats such as sheet music. Porting the platform to other operating systems such as Mac OS and Linux would further expand the target demographic of this annotation tool, as would increasing user friendliness by adding a help menu, tool tips, and comprehensive documentation.

With regards to the research-related matters, one of the more significant insights that unfortunately was out of the scope of this project is to provide a more direct comparison in agreement and time taken to annotate between annotations done using the tool and annotations done using the traditional pen-and-paper approach. While the data generated from ANOMIC's user study was indeed compared to the data in the HEMAN data set, the difference in annotation tasks made comparison difficult. A more controlled, timed experiment where groups of participants annotate using sheet music and ANOMIC on the

same excerpts under the same conditions would be highly beneficial in providing a more concrete and definitive comparison between the two methods of annotation.

Another idea that was unfortunately not explored was to use the annotations generated by the user study in an MIR task such as classification or compression to see how they perform. This is a popular method of measuring the quality of annotations, because in the end, the goal of discovering musical patterns is to facilitate their application in MIR. A strong indicator that annotations generated by the tool are of a desirable quality would be to test their performance in practice.

Finally, the goal of the annotation tool is to be scalable, so conducting more user studies and gathering more annotation data will certainly provide great insight into the annotation tool's performance and usability, along with generating more data to be analysed and used in data-driven pattern discovery algorithms.

5.2 Conclusion

The central point of this research was the development of a novel annotation tool designed for musicologists, researchers, and enthusiasts to annotate patterns in music. Our working definition of a pattern is a segment of notes that repeats exactly or inexactly within the piece. Every repetition of a pattern is considered an occurrence, and thus, a pattern can be seen as an abstract term encompassing a set of occurrences that are similarly or exactly repeated. The developed tool, ANOMIC, offers the following features:

- An interactive, responsive, and mildly customisable interface.
- The ability to annotate any number of patterns, any number of occurrences per pattern, and any number of notes per occurrence.
- The ability to visualise patterns, occurrences, and notes on the interface through colour coding, note highlighting, scrolling, expanding/collapsing, zooming, a grid, and an intuitive piano roll interface.
- The ability to manage occurrences and patterns by adding and removing them, showing or hiding them in the interface, or adjusting a confidence level for whichever occurrence the user pleases.
- Audio playback in the form of media buttons and an interactive seeker whose position can be adjusted along the timeline.
- Support for loading MIDI files into the tool for annotation.
- Multi-channel support allowing users to view and play different MIDI channels separately.
- The option to save and load annotations for a particular MIDI file in the popular JAMS format.

- Optional user action logging for analysis into how users are using the annotation tool.
- An automatic occurrence finder that can find exact matches of a given occurrence within a piece. These matches are robust to chromatic transposition and polyphony.
- Open-source and freely available with no installation required.

The tool was evaluated through a user study with 26 participants of varying musical backgrounds and demographics. Each participant was given six excerpts of classical music as MIDI files and was asked to annotate all six of them to the best of their ability. Their actions were logged and timed, and the annotations they generated were saved as JAMS files and compiled into a data set of 156 files containing 2,763 occurrences across 788 patterns. After annotation, all 26 participants were asked to fill in a short survey where they are able to give their opinions on their experience using the tool.

An analysis into the survey responses was performed, and it was found that the reception from the participants was highly positive. 22 out of the 26 participants stated that they would not prefer traditional pen-and-paper approaches to annotation over the annotation tool, and 17 of those 22 explicitly made it clear that they would prefer to use the annotation tool for musical pattern annotation. The tool was praised for its ease of use, clean and appealing interface, and quality-of-life features such as audio playback and automatic occurrence finding. These findings suggest that the tool is engaging to users and that there is potential for the tool to be used in gamification projects or learning environments.

The annotations themselves were also analysed using two different agreement metrics. Using an agreement metric based on time intervals, we can conclude that inter-annotator agreement improves upon previous work that employed a similar (but not identical) annotation task. Furthermore, it was shown that people with richer musical backgrounds are able to use the tool to greater effect, as they achieved on average 16% higher inter-annotator agreement than the less musically inclined participants using our pattern agreement metrics. Lastly, one of the key features of the tool, an automatic pattern discovery feature, was shown to improve inter-annotator agreement further, with one caveat being that annotators must clearly be made aware of the purposes of that feature for the task at hand so as not to misuse it.

With the novel annotation tool software developed and evaluated, the hope is now that it provides a digital infrastructure for musical pattern annotation, a process that has traditionally involved pen-and-paper approaches with long and arduous manual digitisation times. With this infrastructure in place, pattern annotation can be done in a more intuitive and efficient manner, facilitating the research of human pattern annotations in music.

Appendix A

User Study Instructions

Hi! I'm currently working on my master's thesis. Part of my research involves a software tool developed for musical pattern annotation, and the goal of my research is to see what kind of patterns participants find in music using this tool, what they think of their experience, and how they think it might compare to annotating music by hand on paper.

I'm looking for people who are willing to take part in a user study. It can be done 100% online and you would require a Windows machine. Experience studying music is appreciated but not required. There's no time limit to getting this done and how long you take might vary, but it shouldn't have to take longer than 20-30 minutes of your time. Here are some step-by-step instructions of what I would like you to do:

1. Download the .zip file in the following link:
   https://tinyurl.com/annotationtoolstudy. Extract it to a folder.

2. You should see two folders: "Tool" and "Midi". In the "Tool" folder, double-clicking on "AnnotationTool.exe" runs the software. No installation is required.

3. Watch this video to understand how to use the tool:
   https://tinyurl.com/annotationtoolvideo

4. In the software, you may browse for the .midi files in the "examples" folder and then click on File > Load Annotations to load their respective .jams files so you can see some examples of patterns being annotated.

5. Once you are ready, load the MIDI files located in the "Midi" folder. Annotate repeated patterns in the music. Patterns are distinct, short musical segments or phrases that are considered to be characteristic to a given piece of music and appear multiple times throughout the piece. Be sure to listen to the music and annotate these patterns and their occurrences using the tool. The occurrences don't need to be exact matches, but they should be closely related (compare this to finding occurrences of a leitmotif in a film soundtrack, for example).

6. For any occurrences where you're not sure whether or not they belong to a pattern, right-click on them to set their confidence level to "2" (unsure) or "1" (highly

uncertain).

7. When you are done, save the annotations to a file (ideally use the same filename as the MIDI file that you annotated – e.g. if you annotated bach1.mid, then save the annotations as bach1.jams). Do the same for all of the six MIDI files.

8. After you've annotated all six files (you should have six .jams files at this point), fill in this survey: https://tinyurl.com/annotationtoolsurvey. Take note of what you write down in the "Identifier/Alias" section – this does not need to be your name if you wish to remain anonymous.

9. Attach your six annotation (.jams) files to an email and send them to s.b.wells@students.uu.nl. Please include your Identifier/Alias in the subject or body of the email.

Feel free to send an email to s.b.wells@students.uu.nl or contact me in any other way if you have any questions about this user study, how your data will be used, or how to run the annotation tool. Thank you for your participation!

Appendix B

User Study Survey

Section 1:

Please fill in the following demographics:

0. I consent that I have been sufficiently informed about this study and that my annotations and survey responses will be anonymously used solely for this research:
    - Yes
    - No
1. Identifier/Alias:
2. Age:
    - 16 or under
    - 17-21
    - 22-28
    - 29-40
    - 41-60
    - 61 or above
3. Gender:
    - Male
    - Female
    - Other
    - Prefer not to say
4. Which of the following statements best describe your musical background?
    - I enjoy listening to music.

- I am a hobbyist/professional music composer.
- I can read sheet music.
- I have experience using piano roll interfaces.
- I possess academic qualifications in music.
- I have academic experience in the field of musicology.
- I have experience annotating musical patterns with traditional "pen and paper" approaches.
- I have experience annotating musical patterns with digital software.
- Other:

5. Please specify what musical instrument you primarily play (leave blank if you do not play any musical instruments):
6. If you do play a musical instrument, how proficient do you consider yourself with it? (1=Beginner, 5=Professional)

Section 2:

Please rate the annotation tool on the following metrics:

7. Learnability; The tool is intuitive and quick to learn. (1=Strongly disagree, 5=Strongly agree)
8. Performance; The tool is easy to use and perform tasks on. (1=Strongly disagree, 5=Strongly agree)
9. Productivity; The tool is practical and good for large-scale pattern annotation. (1=Strongly disagree, 5=Strongly agree)
10. Accessibility; The tool is usable by a broad variety of people. (1=Strongly disagree, 5=Strongly agree)
11. User friendliness; The tool is friendly towards users and robust to errors or mistakes. (1=Strongly disagree, 5=Strongly agree)
12. Information scent; The tool's interface is complete and has all the necessary functionality in an easily reachable area in the interface. (1=Strongly disagree, 5=Strongly agree)
13. Flow; The tool flows well from one annotation task to another and does not suffer from workflow interruptions. (1=Strongly disagree, 5= Strongly agree)
14. Customisation; The tool and its interface can be customised to the user's liking. (1=Strongly disagree, 5=Strongly agree)

15. User engagement; The tool is fun and stimulating to use for pattern annotation. (1=Strongly disagree, 5=Strongly agree)
16. Which three aspects of the tool do you think could be improved the most?
17. Which three aspects of the tool do you like the most?

Section 3:

Please respond to the following questions related to your experience using the tool:

18. Did you use the "Find Exact Occurrences" feature in the tool? If so, did you find it helpful?
19. Were there any features that you felt were missing from the tool?
20. How much would you prefer using this tool to annotating using the traditional method (circling or highlighting notes in physical sheet music)? (1=I clearly prefer the traditional method., 5=I clearly prefer this tool.)
21. Why?
22. Do you have any other comments about this tool or the study?

Appendix C

Musical Terminology

Term	Description
Pitch	Sounds are combinations of sound waves that oscillate at particular frequencies. If this amalgamation of frequencies has an identifiable tone, then the sound is said to have a pitch. This perceived quality allows us to determine how high or low the frequencies of a sound are.
Duration	In music, a duration is implied to be the time during which a sound is audible.
Note	A musical note is a representation of a sound that has a duration and a pitch associated with it. When writing music down in sheet format, durations are often represented as whole notes (*semibreves*), half notes (*minims*), quarter notes (*crotchets*), eighth notes (*quavers*), etc. Pitches are often represented using note names, and the naming scheme used in the paper will be the widespread alphabetical format (i.e. A, B, C, etc).
Tempo	The tempo of music indicates how quickly or slowly the music should play. Traditionally, key words such as *Moderato* or *Allegro* would be used to indicate speed, but in the digital domain and for our purposes, it is always measured in BPM (Beats Per Minute).
Beat	A beat is a rhythmic unit of a piece of music whose duration is dictated by the tempo. Beats in music are often audibly accentuated such that even casual listeners are able to identify and clap to the beats of a piece of music.
Bar	Also called a measure, a bar is a method of organising music into short sections that conform to the music's time signature.
Time Signature	The time signature of a piece of music is usually represented as two numbers placed vertically together, with the top number indicating how many beats there are present in a bar and the bottom number indicating how many beats would fit in the duration of a whole note.

Voice	In a musical context, a voice (or part) typically refers to a constituent, continuous part of the music that can be perceived as independent from other notes being sounded at the same time. Often, different voices in a piece of music are played by different instruments, or, say, different hands of a pianist.
Polyphony	Most music will not consist of only one voice. When multiple voices are playing together, there are many different musical terms to describe this that depend on the *melodic interest* of each voice. For instance, one melodically significant voice alongside accompaniment is typically referred to as *monody*. One melody played by multiple voices that are transposed from one another is referred to as *homophony*. *Polyphony*, on the other hand, specifically refers to multiple independent voices, each with melodic interest/significance, playing together simultaneously. That being said, for the scope of this research, our working definition of polyphony simply implies "multiple voices playing together", and they need not necessarily have independent melodic interest.
Melody	A melody is a sequence of notes and pauses (rests) that together form a complete musical thought.
Chord	A chord is a set of simultaneous notes that are sounded together, usually forming part of a song's harmony.
Harmony	The harmony of a piece of music is the structure and composition of the chords that are present within it.
Semitone	In Western music, pitch is typically discretised into semitones, where one semitone is the minimum pitch disparity between two notes.
Interval	The interval of a pair of notes describes the difference in pitch between those two notes (often described in terms of the number of semitones between the notes). The interval of 12 semitones (or 8 notes) is defined to be an *octave*, and the letter names of two notes one octave apart are equal. Because humans perceive pitch logarithmically, going up in pitch by one octave doubles the sound's frequency, and going down in pitch by one octave halves it.
Accidental	An accidental comes in the form of a flat, sharp, or natural sign accompanying a note. A flat indicates that the note in question is to be played one semitone below its regular pitch. A sharp indicates that the note in question is to be played one semitone above its regular pitch. A natural indicates that any sharp or flat accidentals that the note would have been under the effect of are to be ignored in favour of the note's unaltered pitch.

Key Signature	The key signature is a collection of flat or sharp symbols placed at specific pitches at the start of a bar, implying that all notes with those pitches beyond that point have the flat or sharp alteration applied to them.
Transposition	The type of transposition that this thesis is concerned with is *chromatic transposition*. A set of notes X is defined to be transposed chromatically from another set of notes Y when every note in X is shifted in pitch by the same interval from its corresponding note in Y.
Dynamics	The dynamics of a piece are related to fluctuations in the loudness of a piece. Traditionally, this would be signified by using symbols in sheet music that signify *crescendo* (gradually increase loudness), *piano* (soft), *mezzo forte* (moderately loud), etc. In symbolic music (such as MIDI), the dynamics of a piece are tied to the *velocities* of its notes[1].

Table C.1: A dictionary of musical terms relevant to this thesis.

[1]Velocities will be discussed in greater detail in **Section 2.2**.

Appendix D

Results of Interval Agreement

D.1 HEMAN Results

D.1.1 bach1.mid

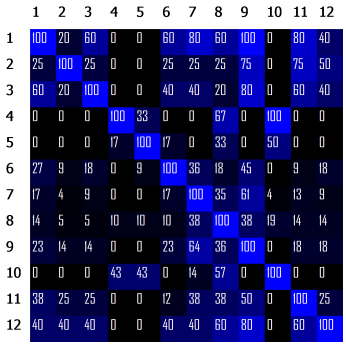


Figure D.1: Precision

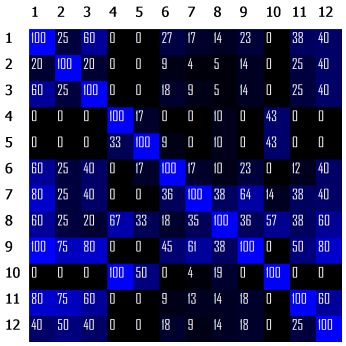


Figure D.2: Recall

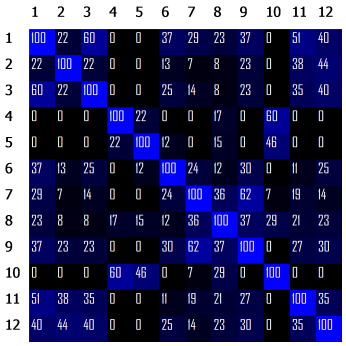


Figure D.3: F-Score

D.1.2 bach2.mid

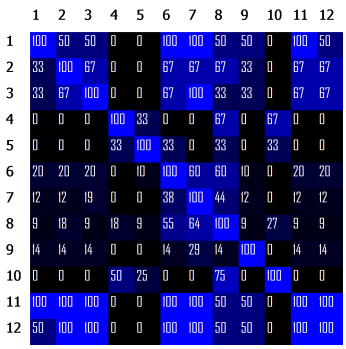


Figure D.4: Precision

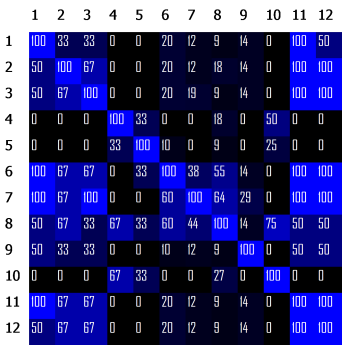


Figure D.5: Recall

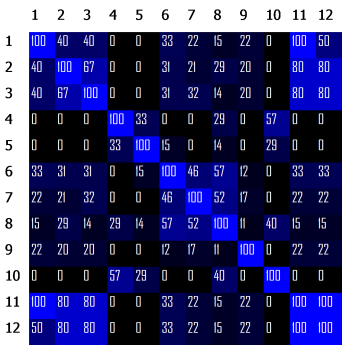


Figure D.6: F-Score

D.1.3 bee1.mid

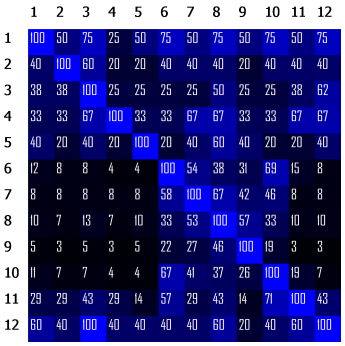


Figure D.7: Precision

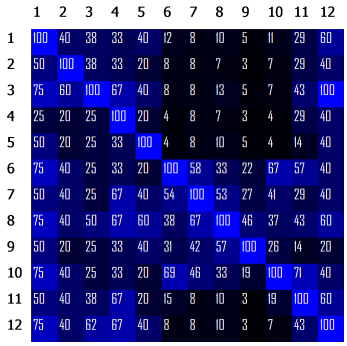


Figure D.8: Recall

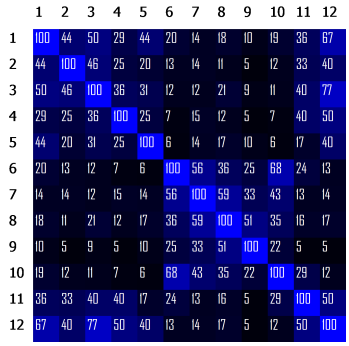


Figure D.9: F-Score

D.1.4 hay1.mid

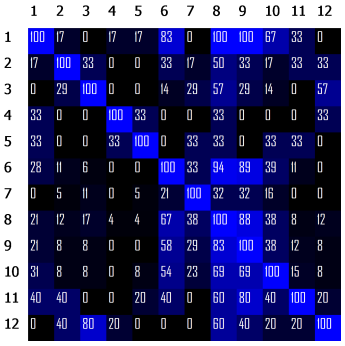


Figure D.10: Precision

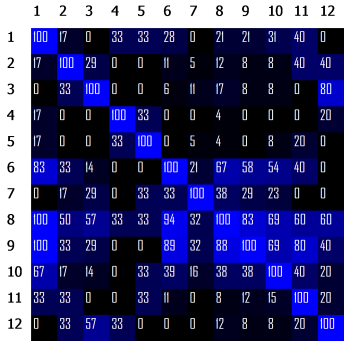


Figure D.11: Recall

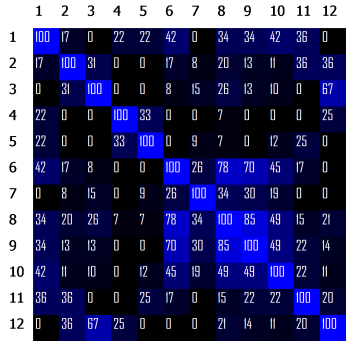


Figure D.12: F-Score

D.1.5 mo155.mid

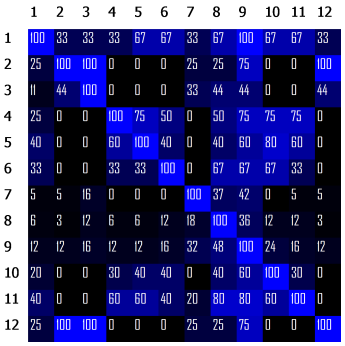


Figure D.13: Precision

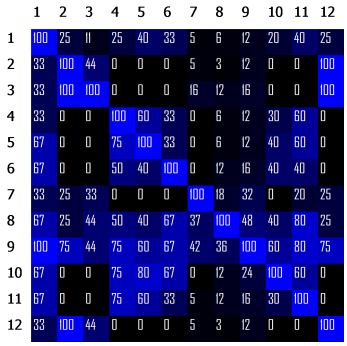


Figure D.14: Recall

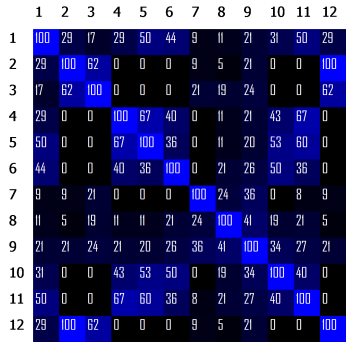


Figure D.15: F-Score

D.1.6 mo458.mid

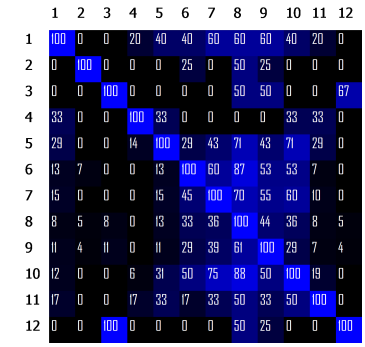


Figure D.16: Precision

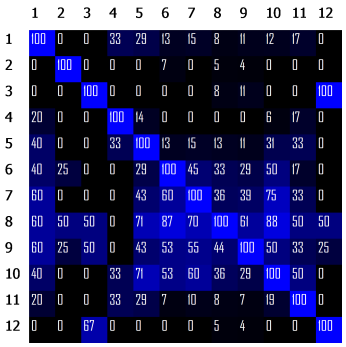


Figure D.17: Recall

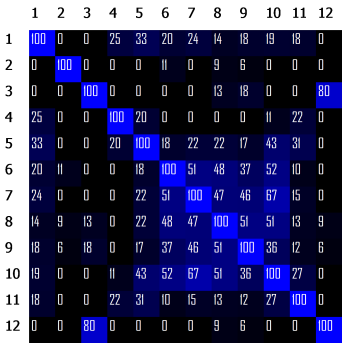


Figure D.18: F-Score

D.2 ANOMIC Results

D.2.1 bach1.mid

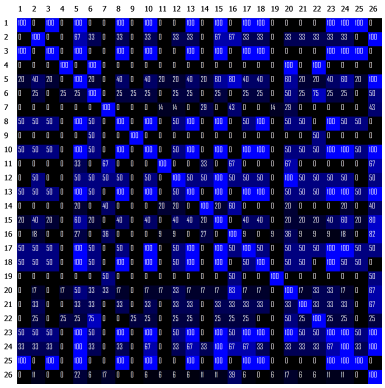


Figure D.19: Precision

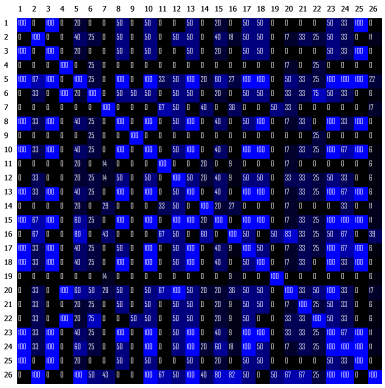


Figure D.20: Recall

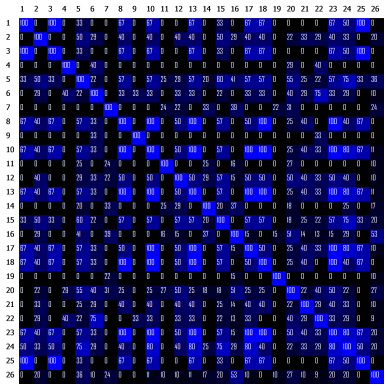


Figure D.21: F-Score

D.2.2 bach2.mid

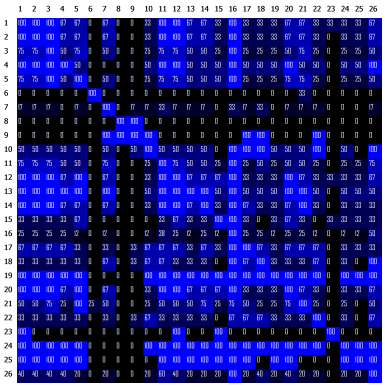


Figure D.22: Precision

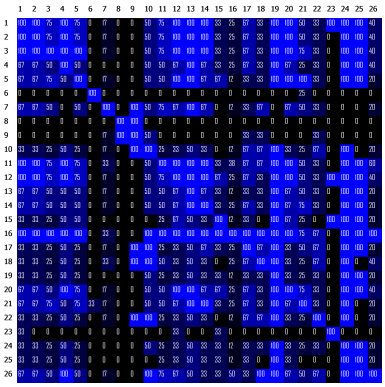


Figure D.23: Recall

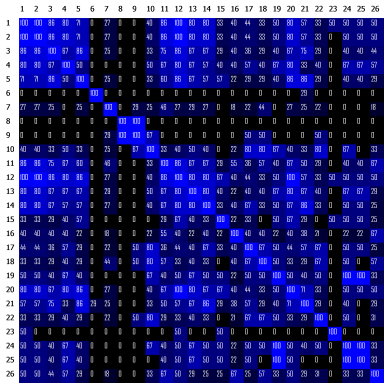


Figure D.24: F-Score

D.2.3 bee1.mid

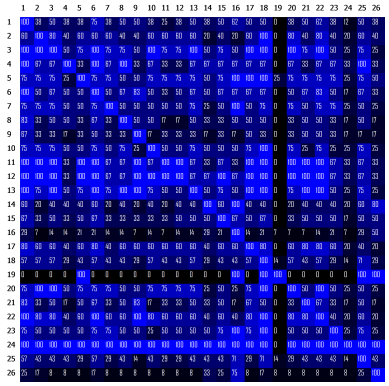


Figure D.25: Precision

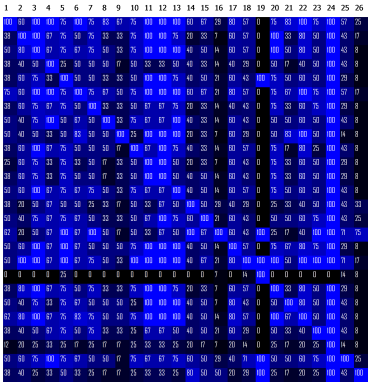


Figure D.26: Recall

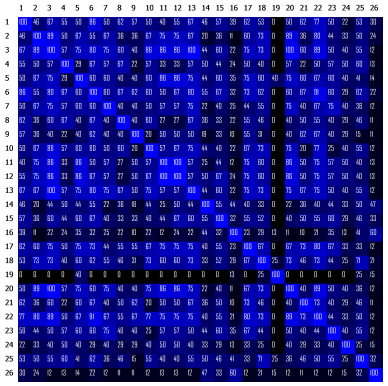


Figure D.27: F-Score

D.2.4 hay1.mid

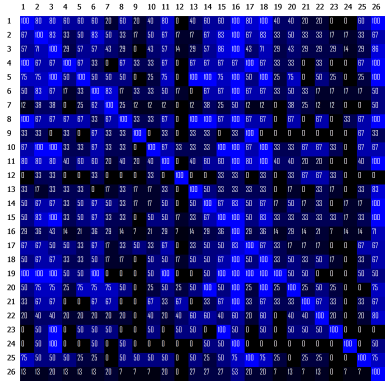


Figure D.28: Precision

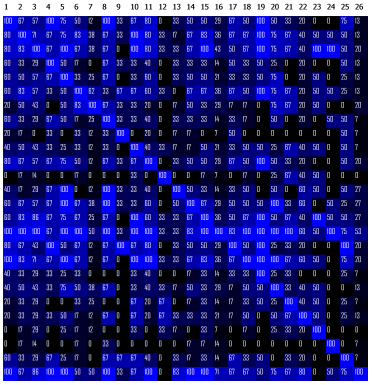


Figure D.29: Recall

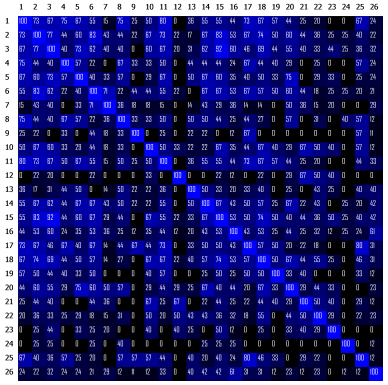


Figure D.30: F-Score

D.2.5 mo155.mid

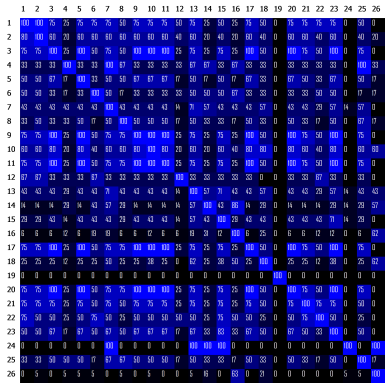


Figure D.31: Precision

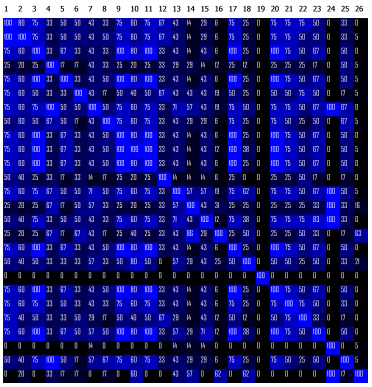


Figure D.32: Recall

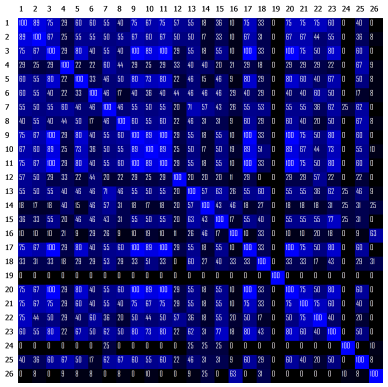


Figure D.33: F-Score

D.2.6 mo458.mid

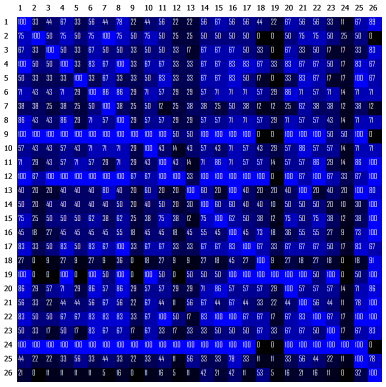


Figure D.34: Precision

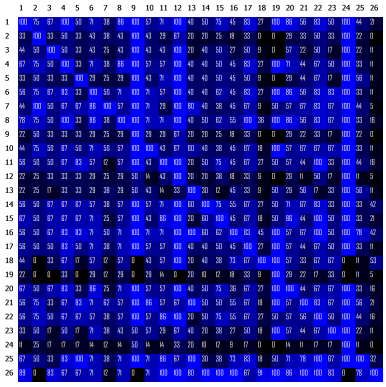


Figure D.35: Recall

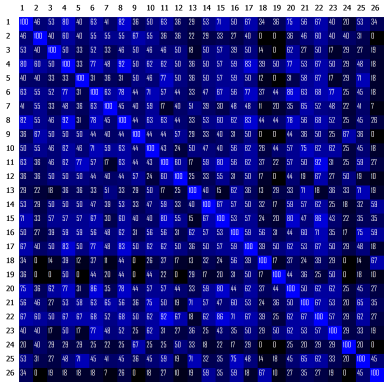


Figure D.36: F-Score

Appendix E

Results of Note-Level Agreement

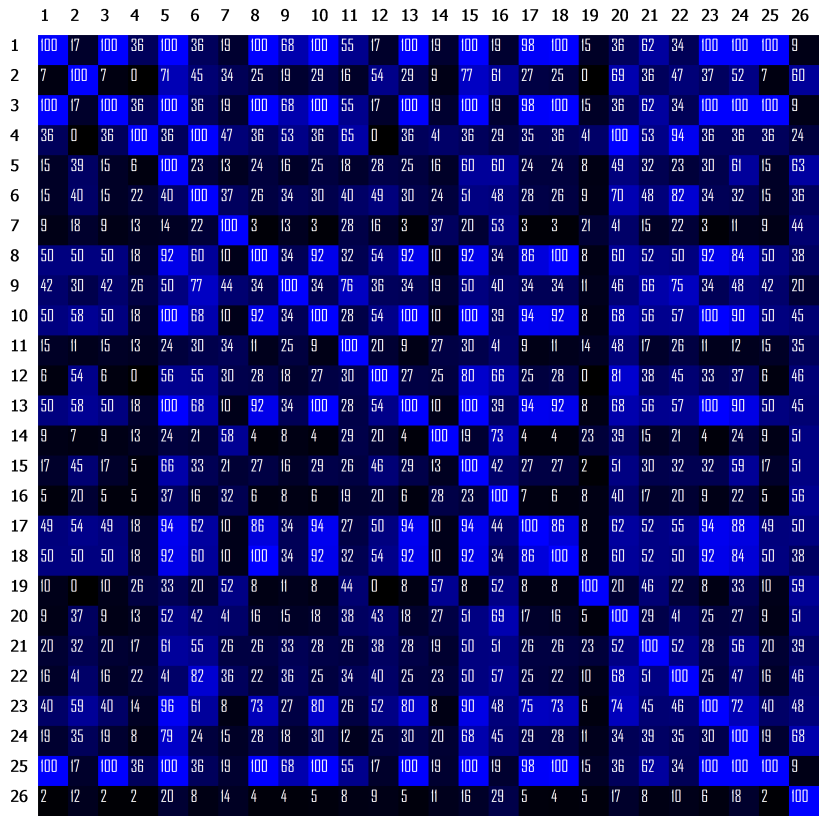
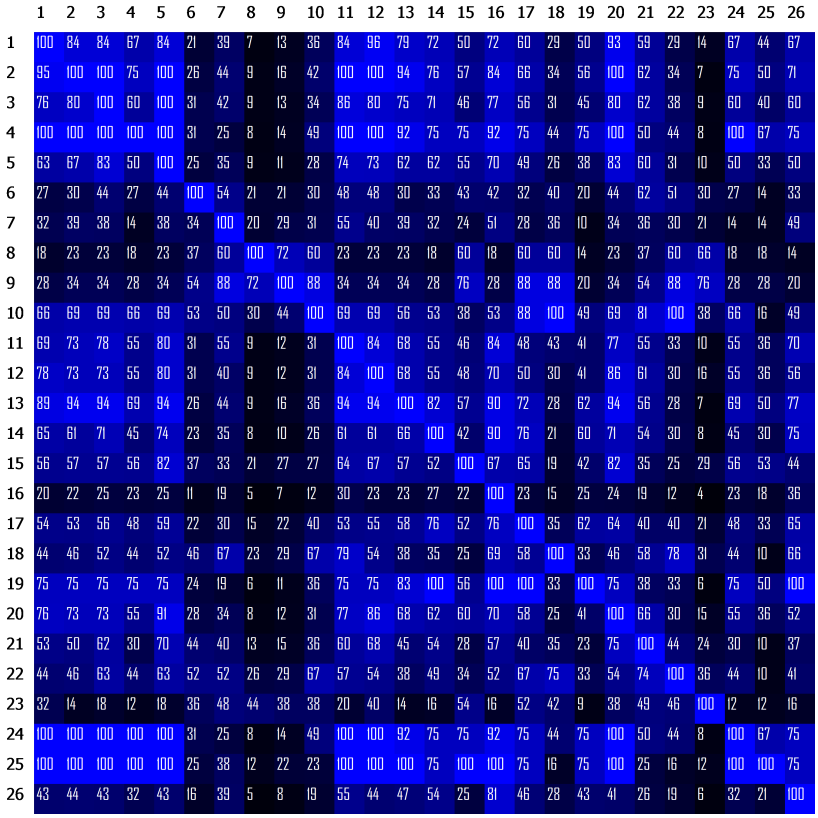
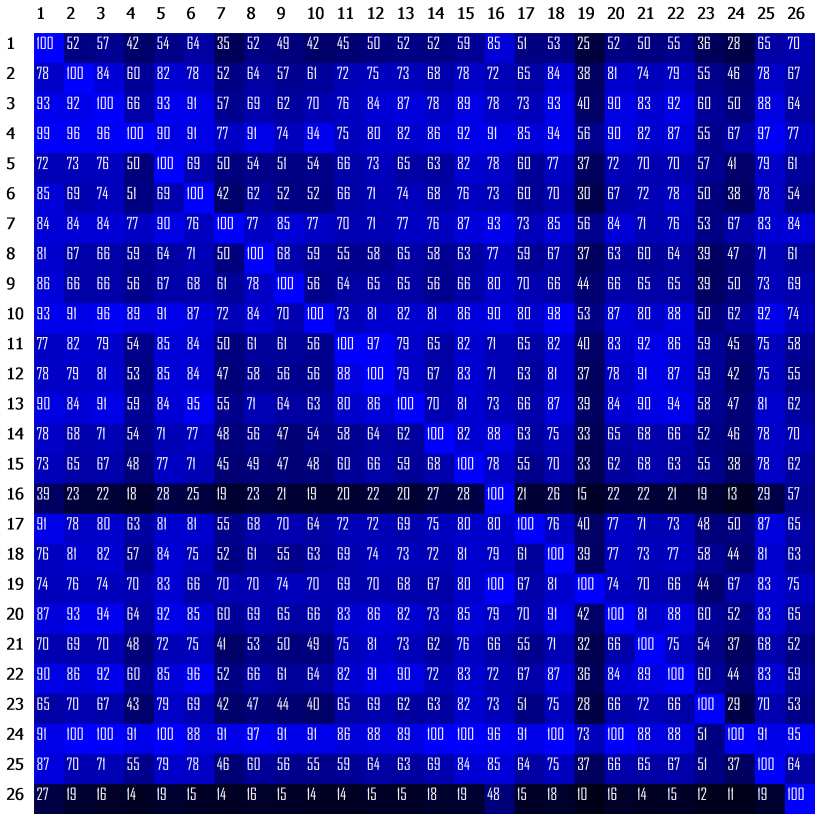


Figure E.1: The agreement matrix for bach1.mid.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
1	100	84	84	67	84	21	39	7	13	36	84	96	79	72	50	72	60	29	50	93	59	29	14	67	44	67
2	95	100	100	75	100	26	44	9	16	42	100	100	94	76	57	84	66	34	56	100	62	34	7	75	50	71
3	76	80	100	60	100	31	42	9	13	34	86	80	75	71	46	77	56	31	45	80	62	38	9	60	40	60
4	100	100	100	100	100	31	25	8	14	49	100	100	92	75	75	92	75	44	75	100	50	44	8	100	67	75
5	63	67	83	50	100	25	35	9	11	28	74	73	62	62	55	70	49	26	38	83	60	31	10	50	33	50
6	27	30	44	27	44	100	54	21	21	30	48	48	30	33	43	42	32	40	20	44	62	51	30	27	14	33
7	32	39	38	14	38	34	100	20	29	31	55	40	39	32	24	51	28	36	10	34	36	30	21	14	14	49
8	18	23	23	18	23	37	60	100	72	60	23	23	23	18	60	18	60	60	14	23	37	60	66	18	18	14
9	28	34	34	28	34	54	88	72	100	88	34	34	34	28	76	28	88	88	20	34	54	88	76	28	28	20
10	66	69	69	66	69	53	50	30	44	100	69	69	56	53	38	53	88	100	49	69	81	100	38	66	16	49
11	69	73	78	55	80	31	55	9	12	31	100	84	68	55	46	84	48	43	41	77	55	33	10	55	36	70
12	78	73	73	55	80	31	40	9	12	31	84	100	68	55	48	70	50	30	41	86	61	30	16	55	36	56
13	89	94	94	69	94	26	44	9	16	36	94	94	100	82	57	90	72	28	62	94	56	28	7	69	50	77
14	65	61	71	45	74	23	35	8	10	26	61	61	66	100	42	90	76	21	60	71	54	30	8	45	30	75
15	56	57	57	56	82	37	33	21	27	27	64	67	57	52	100	67	65	19	42	82	35	25	29	56	53	44
16	20	22	25	23	25	11	19	5	7	12	30	23	23	27	22	100	23	15	25	24	19	12	4	23	18	36
17	54	53	56	48	59	22	30	15	22	40	53	55	58	76	52	76	100	35	62	64	40	40	21	48	33	65
18	44	46	52	44	52	46	67	23	29	67	79	54	38	35	25	69	58	100	33	46	58	78	31	44	10	66
19	75	75	75	75	75	24	19	6	11	36	75	75	83	100	56	100	100	33	100	75	38	33	6	75	50	100
20	76	73	73	55	91	28	34	8	12	31	77	86	68	62	60	70	58	25	41	100	66	30	15	55	36	52
21	53	50	62	30	70	44	40	13	15	36	60	68	45	54	28	57	40	35	23	75	100	44	24	30	10	37
22	44	46	63	44	63	52	52	26	29	67	57	54	38	49	34	52	67	75	33	54	74	100	36	44	10	41
23	32	14	18	12	18	36	48	44	38	38	20	40	14	16	54	16	52	42	9	38	49	46	100	12	12	16
24	100	100	100	100	100	31	25	8	14	49	100	100	92	75	75	92	75	44	75	100	50	44	8	100	67	75
25	100	100	100	100	100	25	38	12	22	23	100	100	100	75	100	100	75	16	75	100	25	16	12	100	100	75
26	43	44	43	32	43	16	39	5	8	19	55	44	47	54	25	81	46	28	43	41	26	19	6	32	21	100

Figure E.2: The agreement matrix for bach2.mid.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
1	100	52	57	42	54	64	35	52	49	42	45	50	52	52	59	85	51	53	25	52	50	55	36	28	65	70
2	78	100	84	60	82	78	52	64	57	61	72	75	73	68	78	72	65	84	38	81	74	79	55	46	78	67
3	93	92	100	66	93	91	57	69	62	70	76	84	87	78	89	78	73	93	40	90	83	92	60	50	88	64
4	99	96	96	100	90	91	77	91	74	94	75	80	82	86	92	91	85	94	56	90	82	87	55	67	97	77
5	72	73	76	50	100	69	50	54	51	54	66	73	65	63	82	78	60	77	37	72	70	70	57	41	79	61
6	85	69	74	51	69	100	42	62	52	52	66	71	74	68	76	73	60	70	30	67	72	78	50	38	78	54
7	84	84	84	77	90	76	100	77	85	77	70	71	77	76	87	93	73	85	56	84	71	76	53	67	83	84
8	81	67	66	59	64	71	50	100	68	59	55	58	65	58	63	77	59	67	37	63	60	64	39	47	71	61
9	86	66	66	56	67	68	61	78	100	56	64	65	65	56	66	80	70	66	44	66	65	65	39	50	73	69
10	93	91	96	89	91	87	72	84	70	100	73	81	82	81	86	90	80	98	53	87	80	88	50	62	92	74
11	77	82	79	54	85	84	50	61	61	56	100	97	79	65	82	71	65	82	40	83	92	86	59	45	75	58
12	78	79	81	53	85	84	47	58	56	56	88	100	79	67	83	71	63	81	37	78	91	87	59	42	75	55
13	90	84	91	59	84	95	55	71	64	63	80	86	100	70	81	73	66	87	39	84	90	94	58	47	81	62
14	78	68	71	54	71	77	48	56	47	54	58	64	62	100	82	88	63	75	33	65	68	66	52	46	78	70
15	73	65	67	48	77	71	45	49	47	48	60	66	59	68	100	78	55	70	33	62	68	63	55	38	78	62
16	39	23	22	18	28	25	19	23	21	19	20	22	20	27	28	100	21	26	15	22	22	21	19	13	29	57
17	91	78	80	63	81	81	55	68	70	64	72	72	69	75	80	80	100	76	40	77	71	73	48	50	87	65
18	76	81	82	57	84	75	52	61	55	63	69	74	73	72	81	79	61	100	39	77	73	77	58	44	81	63
19	74	76	74	70	83	66	70	70	74	70	69	70	68	67	80	100	67	81	100	74	70	66	44	67	83	75
20	87	93	94	64	92	85	60	69	65	66	83	86	82	73	85	79	70	91	42	100	81	88	60	52	83	65
21	70	69	70	48	72	75	41	53	50	49	75	81	73	62	76	66	55	71	32	66	100	75	54	37	68	52
22	90	86	92	60	85	96	52	66	61	64	82	91	90	72	83	72	67	87	36	84	89	100	60	44	83	59
23	65	70	67	43	79	69	42	47	44	40	65	69	62	63	82	73	51	75	28	66	72	66	100	29	70	53
24	91	100	100	91	100	88	91	97	91	91	86	88	89	100	100	96	91	100	73	100	88	88	51	100	91	95
25	87	70	71	55	79	78	46	60	56	55	59	64	63	69	84	85	64	75	37	66	65	67	51	37	100	64
26	27	19	16	14	19	15	14	16	15	14	14	15	15	18	19	48	15	18	10	16	14	15	12	11	19	100

Figure E.3: The agreement matrix for bee1.mid.

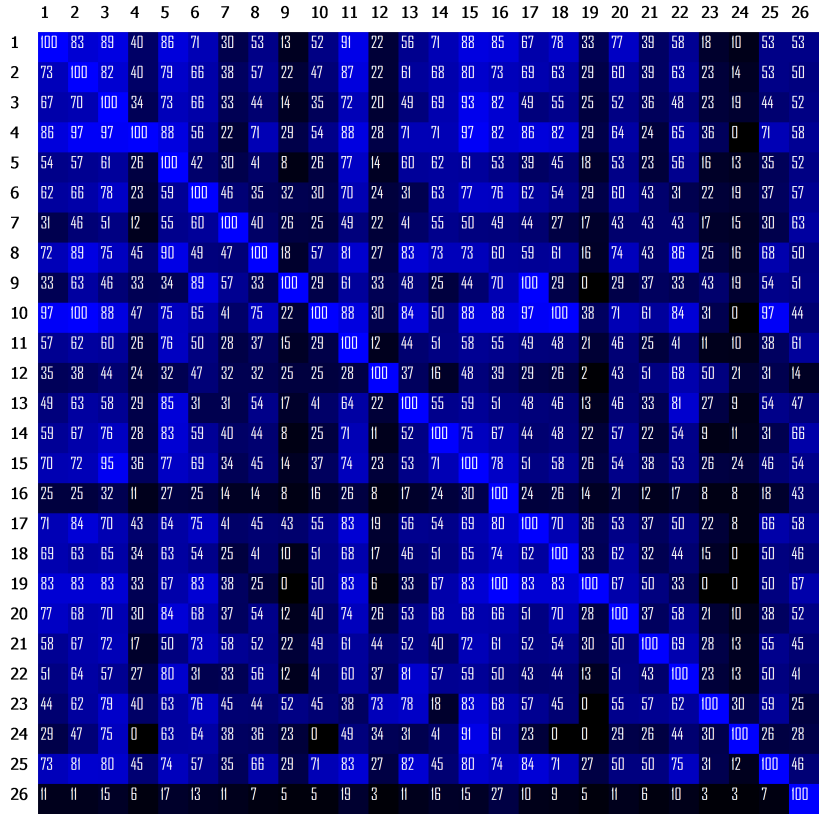


Figure E.4: The agreement matrix for hay1.mid.

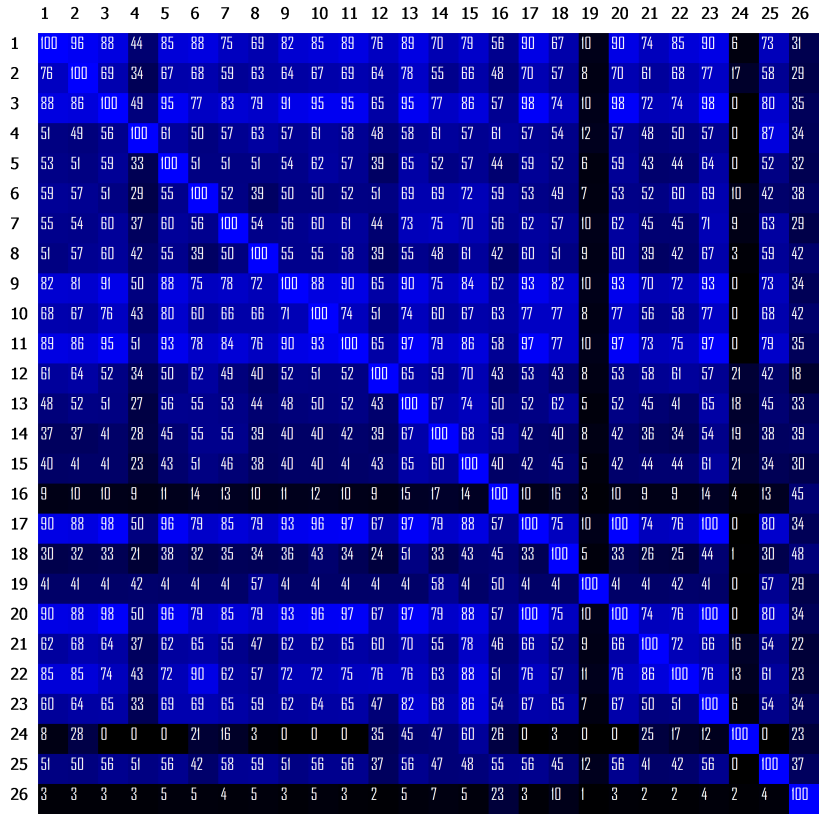
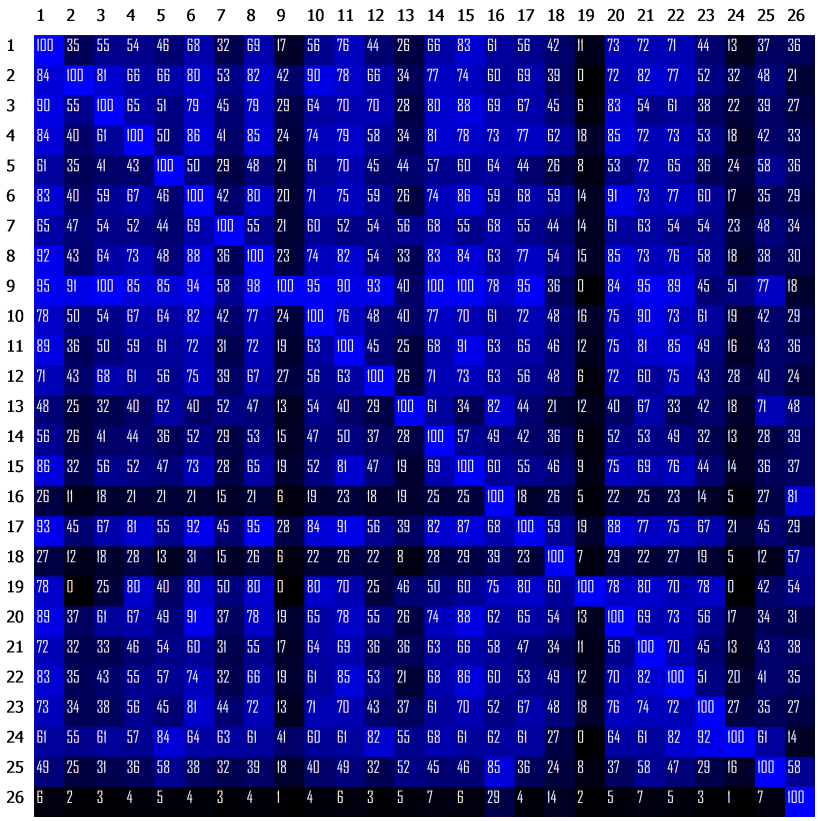


Figure E.5: The agreement matrix for mo155.mid.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
1	100	35	55	54	46	68	32	69	17	56	76	44	26	66	83	61	56	42	11	73	72	71	44	13	37	36
2	84	100	81	66	66	80	53	82	42	90	78	66	34	77	74	60	69	39	0	72	82	77	52	32	48	21
3	90	55	100	65	51	79	45	79	29	64	70	70	28	80	88	69	67	45	6	83	54	61	38	22	39	27
4	84	40	61	100	50	86	41	85	24	74	79	58	34	81	78	73	77	62	18	85	72	73	53	18	42	33
5	61	35	41	43	100	50	29	48	21	61	70	45	44	57	60	64	44	26	8	53	72	65	36	24	58	36
6	83	40	59	67	46	100	42	80	20	71	75	59	26	74	86	59	68	59	14	91	73	77	60	17	35	29
7	65	47	54	52	44	69	100	55	21	60	52	54	56	68	55	68	55	44	14	61	63	54	54	23	48	34
8	92	43	64	73	48	88	36	100	23	74	82	54	33	83	84	63	77	54	15	85	73	76	58	18	38	30
9	95	91	100	85	85	94	58	98	100	95	90	93	40	100	100	78	95	36	0	84	95	89	45	51	77	18
10	78	50	54	67	64	82	42	77	24	100	76	48	40	77	70	61	72	48	16	75	90	73	61	19	42	29
11	89	36	50	59	61	72	31	72	19	63	100	45	25	68	91	63	65	46	12	75	81	85	49	16	43	36
12	71	43	68	61	56	75	39	67	27	56	63	100	26	71	73	63	56	48	6	72	60	75	43	28	40	24
13	48	25	32	40	62	40	52	47	13	54	40	29	100	61	34	82	44	21	12	40	67	33	42	18	71	48
14	56	26	41	44	36	52	29	53	15	47	50	37	28	100	57	49	42	36	6	52	53	49	32	13	28	39
15	86	32	56	52	47	73	28	65	19	52	81	47	19	69	100	60	55	46	9	75	69	76	44	14	36	37
16	26	11	18	21	21	21	15	21	6	19	23	18	19	25	25	100	18	26	5	22	25	23	14	5	27	81
17	93	45	67	81	55	92	45	95	28	84	91	56	39	82	87	68	100	59	19	88	77	75	67	21	45	29
18	27	12	18	28	13	31	15	26	6	22	26	22	8	28	29	39	23	100	7	29	22	27	19	5	12	57
19	78	0	25	80	40	80	50	80	0	80	70	25	46	50	60	75	80	60	100	78	80	70	78	0	42	54
20	89	37	61	67	49	91	37	78	19	65	78	55	26	74	88	62	65	54	13	100	69	73	56	17	34	31
21	72	32	33	46	54	60	31	55	17	64	69	36	36	63	66	58	47	34	11	56	100	70	45	13	43	38
22	83	35	43	55	57	74	32	66	19	61	85	53	21	68	86	60	53	49	12	70	82	100	51	20	41	35
23	73	34	38	56	45	81	44	72	13	71	70	43	37	61	70	52	67	48	18	76	74	72	100	27	35	27
24	61	55	61	57	84	64	63	61	41	60	61	82	55	68	61	62	61	27	0	64	61	82	92	100	61	14
25	49	25	31	36	58	38	32	39	18	40	49	32	52	45	46	85	36	24	8	37	58	47	29	16	100	58
26	6	2	3	4	5	4	3	4	1	4	6	3	5	7	6	29	4	14	2	5	7	5	3	1	7	100

Figure E.6: The agreement matrix for mo458.mid.

Bibliography

Audacity. Retrieved from https://www.audacityteam.org

Avid - Sibelius. Retrieved from https://www.avid.com/sibelius-ultimate/

Bach, J. S. (1720). Sinfonia No. 1 in C Major, BWV 787. *Inventions and Sinfonias, BWV 772–801*.

Bach, J. S. (1725). Cantata BWV 1, Movement 6, Horn. *Wie schön leuchtet der Morgenstern*.

Bayard, S. P. (1950). Prolegomena to a study of the principal melodic families of British-American folk song. *The Journal of American Folklore*, *63*(247), 1–44.

Bent, I. & Drabkin, W. (1987). *Analysis*. New Grove handbooks in music. Macmillan.

Black, H. S. & Edson, J. (1947). Pulse code modulation. *Transactions of the American Institute of Electrical Engineers*, *66*(1), 895–899.

Blue - Yeti. Retrieved from https://www.bluedesigns.com/products/yeti/

Boot, P., Volk, A., & De Haas, W. B. (2016). Evaluating the role of repeated patterns in folk song classification and compression. *Journal of New Music Research*, *45*(3), 223–238.

Bribitzer-Stull, M. (2015). Introduction: The leitmotif problem. In *Understanding the leitmotif* (pp. 7–10). Cambridge University Press.

C++ Programming/Code/API/Win32. Retrieved from https://docs.microsoft.com/en-us/windows/win32/apiindex/windows-api-list

Cannam, C., Landone, C., & Sandler, M. (2010). Sonic visualiser: An open source application for viewing, analysing, and annotating music audio files. In *Proceedings of the 18th ACM international conference on multimedia* (pp. 1467–1468). ACM.

Carletta, J., Ashby, S., Bourban, S., Flynn, M., Guillemot, M., Hain, T., ... Wellner, P. (2005). The AMI meeting corpus: A pre-announcement. In *International Workshop on Machine Learning for Multimodal Interaction* (pp. 28–39). Springer.

Collins, T. (2019). 2019:Discovery of repeated themes & sections. Accessed: 2019-06-06. Retrieved from https://www.music-ir.org/mirex/wiki/2019:Discovery_of_Repeated_Themes_&_Sections

Collins, T., Laney, R., Willis, A., & Garthwaite, P. H. (2011). Modeling pattern importance in Chopin's mazurkas. *Music Perception: An Interdisciplinary Journal*, *28*(4), 387–414.

Deutsch, D. & Feroe, J. (1981). The internal representation of pitch sequences in tonal music. *Psychological review*, *88*(6), 503.

DiCarlo, J. J., Zoccolan, D., & Rust, N. C. (2012). How does the brain solve visual object recognition? *Neuron*, *73*(3), 415–434.

Feldman, J. (1961). Simulation of behavior in the binary choice experiment. In *Papers presented at the May 9-11, 1961, western joint IRE-AIEE-ACM computer conference* (pp. 133–144). ACM.

Finale Notepad. Retrieved from https://www.finalemusic.com/products/notepad/

FL Studio. Retrieved from https://www.image-line.com/flstudio/

GitHub - irisyupingren/HEMANanalysis. Retrieved from https://github.com/irisyupingren/HEMANanalysis

GitHub - JKUPDD-Aug2013. Retrieved from https://github.com/wangsix/VMO_repeated_themes_discovery/tree/master/JKUPDD-Aug2013

GitHub - jstnryan/midi-dot-net. Retrieved from https://github.com/jstnryan/midi-dot-net

Google Docs. Retrieved from https://www.google.com/docs/about/

Google Forms. Retrieved from https://www.google.com/forms/about/

Goutte, C. & Gaussier, E. (2005). A probabilistic interpretation of precision, recall and F-score, with implication for evaluation. In *European Conference on Information Retrieval* (pp. 345–359). Springer.

Gowrishankar, B. & Bhajantri, N. U. (2016). An exhaustive review of automatic music transcription techniques: Survey of music transcription techniques. In *2016 International Conference on Signal Processing, Communication, Power and Embedded System (SCOPES)* (pp. 140–152). IEEE.

Hainsworth, S. W. & Macleod, M. D. (2003). Onset detection in musical audio signals. In *ICMC*.

Herbert, S. A. & Sumner, R. K. (1993). *Pattern in music*. MIT Press.

Hsu, J.-L., Liu, C.-C., & Chen, A. L. (2001). Discovering nontrivial repeating patterns in music data. *IEEE Transactions on Multimedia*, *3*(3), 311–325.

Huber, D. M. (2007). *The MIDI manual: A practical guide to midi in the project studio*. Taylor & Francis.

Humphrey, E. J., Salamon, J., Nieto, O., Forsyth, J., Bittner, R. M., & Bello, J. P. (2014). JAMS: A JSON annotated music specification for reproducible MIR research. In *Proceedings of the 15th International Society for Music Information Retrieval Conference (ISMIR 2014)* (pp. 591–596).

Hyatt King, A. (1964). *Four hundred years of music printing*. Trustees of the British Museum.

Jain, A. K., Duin, R. P. W., & Mao, J. (2000). Statistical pattern recognition: A review. *IEEE Transactions on pattern analysis and machine intelligence*, *22*(1), 4–37.

JAMS Structure - jams 0.3.2 documentation. Retrieved from https://jams.readthedocs.io/en/stable/index.html

Janssen, B., De Haas, W. B., Volk, A., & Van Kranenburg, P. (2013). Discovering repeated patterns in music: State of knowledge, challenges, perspectives. In *Proceedings of the 10th International Symposium on Computer Music Multidisciplinary Research, Marseille, France* (Vol. 20, p. 74).

Krumhansl, C. L. & Jusczyk, P. W. (1990). Infants' perception of phrase structure in music. *Psychological science*, *1*(1), 70–73.

Lattner, S., Grachten, M., & Widmer, G. (2018). Learning transposition-invariant interval features from symbolic music and audio. In *Proceedings of the 19th ISMIR Conference, Paris, France*.

Law, E. L., Von Ahn, L., Dannenberg, R. B., & Crawford, M. (2007). TagATune: A game for music and sound annotation. In *ISMIR* (Vol. 3, p. 2).

MEGA. Retrieved from https://mega.nz/

Melkonian, O., Ren, I. Y., Swierstra, W., & Volk, A. (2019). What constitutes a musical pattern? In *Proceedings of the 7th ACM SIGPLAN International Workshop on Functional Art, Music, Modeling, and Design (FARM '19), August 23, 2019, Berlin, Germany* (pp. 95–105).

Meredith, D. (2006). The ps13 pitch spelling algorithm. *Journal of New Music Research*, *35*(2), 121–159.

Meredith, D. (2015). Music analysis and point-set compression. *Journal of New Music Research*, *44*(3), 245–270.

Meredith, D., Lemström, K., & Wiggins, G. A. (2002). Algorithms for discovering repeated patterns in multidimensional representations of polyphonic music. *Journal of New Music Research*, *31*(4), 321–345.

Microsoft Office. Retrieved from https://www.microsoft.com/en-au/download/office.aspx

Midi Sheet Music. Retrieved from http://midisheetmusic.com/

Moog, R. A. (1986). MIDI: Musical instrument digital interface. *Journal of the Audio Engineering Society*, *34*(5), 394–404.

Movavi Video Editor. Retrieved from https://www.movavi.com/videoeditor/

MuseScore. Retrieved from https://musescore.org/en

NVIDIA ShadowPlay. Retrieved from https://www.nvidia.com/en-us/geforce/geforce-experience/shadowplay/

Poliner, G. E., Ellis, D. P., Ehmann, A. F., Gómez, E., Streich, S., & Ong, B. (2007). Melody transcription from music audio: Approaches and evaluation. *IEEE Transactions on Audio, Speech, and Language Processing*, *15*(4), 1247–1256.

Reidsma, D. & Op Den Akker, R. (2008). Exploiting 'subjective' annotations. In *Proceedings of the Workshop on Human Judgements in Computational Linguistics* (pp. 8–16). Association for Computational Linguistics.

Ren, I. Y., Koops, H. V., Volk, A., & Swierstra, W. (2017). In search of the consensus among musical pattern discovery algorithms. In *Proceedings of the 18th International Society for Music Information Retrieval Conference* (pp. 671–678). ISMIR press.

Ren, I. Y., Nieto, O., Koops, H. V., Volk, A., & Swierstra, W. (2018). Investigating musical pattern ambiguity in a human annotated dataset. In *Proceedings of the 15th International Conference on Music Perception and Cognition/10th triennial conference of the European Society for the Cognitive Sciences of Music*.

Ren, I. Y., Volk, A., Swierstra, W., & Veltkamp, R. C. (2018). Analysis by classification: A comparative study of annotated and algorithmically extracted patterns in symbolic music data. In *Proceedings of the 19th ISMIR Conference, Paris, France* (pp. 661–667).

Russell, B. C., Torralba, A., Murphy, K. P., & Freeman, W. T. (2008). LabelMe: A database and web-based tool for image annotation. *International journal of computer vision*, *77*(1-3), 157–173.

Saint-Saëns, C. (1872). Danse Macabre. Op. 40.

The Sonic Spot - MIDI File Format. Retrieved from https://web.archive.org/web/20141227205754/http://www.sonicspot.com:80/guide/midifiles.html

Turnbull, D., Liu, R., Barrington, L., & Lanckriet, G. R. (2007). A game-based approach for collecting semantic annotations of music. In *Proceedings of the 8th ISMIR* (pp. 535–538).

Visual Studio. Retrieved from https://visualstudio.microsoft.com/downloads/

Volk, A. & Van Kranenburg, P. (2012). Melodic similarity among folk songs: An annotation study on similarity-based categorization in music. *Musicae Scientiae*, *16*(3), 317–339.

Zins, C. (2007). Conceptual approaches for defining data, information, and knowledge. *Journal of the American society for information science and technology*, *58*(4), 479–493.