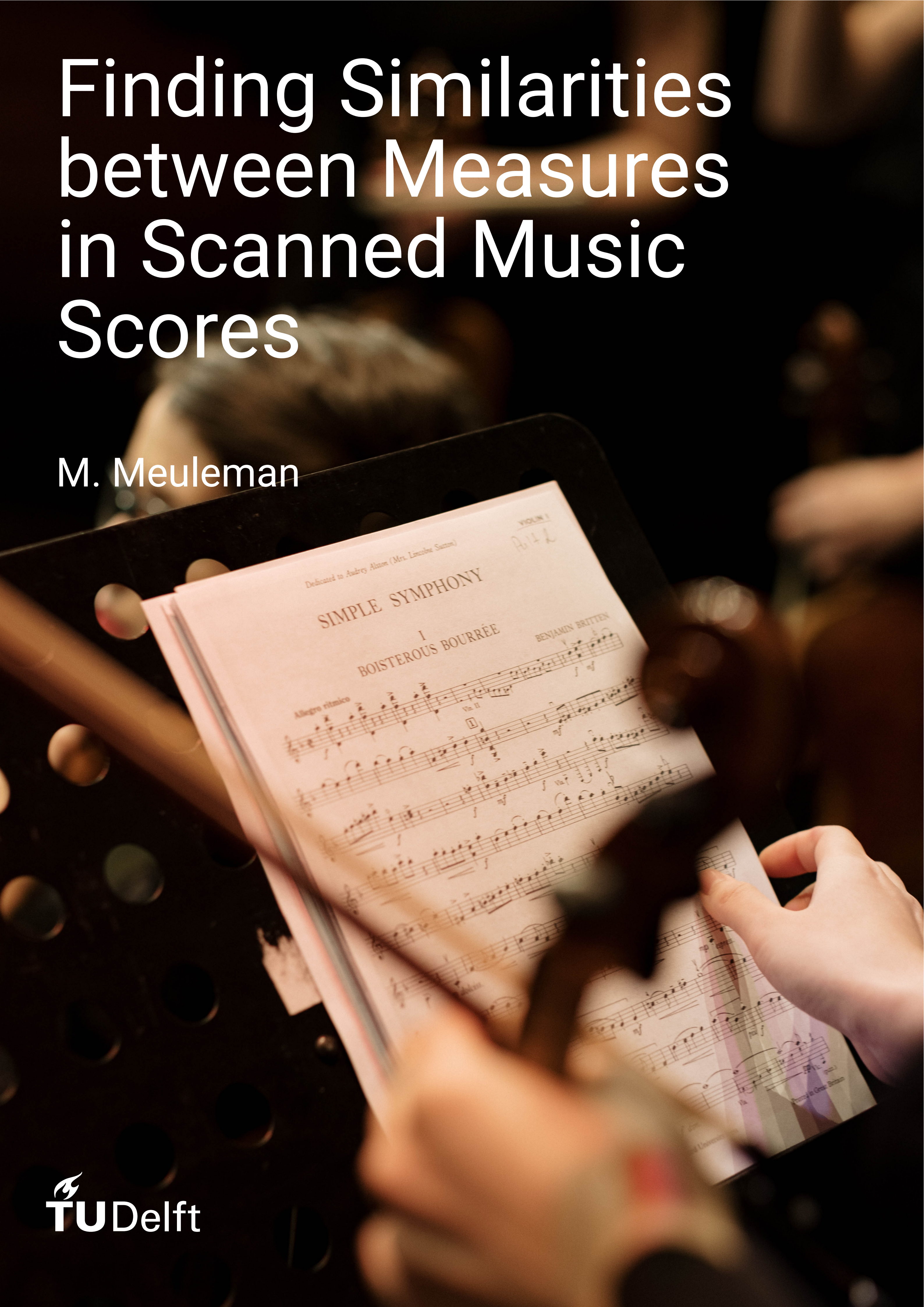
Finding Similarities between Measures in Scanned Music Scores

M. Meuleman



TU Delft

This page was intentionally left blank.

Finding Similarities between Measures in Scanned Music Scores

by

M. Meuleman

to obtain the degree of Master of Science

at the Delft University of Technology,

to be defended publicly on Monday December 20, 2021 at 01:00 PM.

Student number:	4375629	
Project duration:	July 1, 2020 – December 20, 2021	
Thesis committee:	Dr. C. C. S. Liem MMus,	TU Delft, supervisor
	Dr. C. Lofi,	TU Delft

TU Delft

This page was intentionally left blank.

Abstract

The field of Optical Music Recognition has been making progress in the past decades to automate the process of transcribing music scores into computer-readable formats, but its results are still far from being generally applicable. Some research effort has focused on incorporating crowdsourcing techniques into this field to check and correct errors in existing methods. However, since this is a costly process, improvements in efficiency are key to making these systems applicable for music transcription. A possible solution is to identify which measures are similar to each other, so that obtained transcriptions can be shared between them.

In this thesis, we propose a system that generates clusters of measures that are segmented from music scores, in order to find measures that are similar to each other. To design and evaluate this system, we have collected an extensive dataset of published music scores and their symbolic transcriptions. We have manually created annotations that map the symbolic scores to the printed scores on the measure level, so that we can obtain a ground truth from it. We have improved an existing staff detection algorithm and compared it to the original using our dataset. We have also developed our own barline detection algorithm and evaluated it against the dataset. We used both these algorithms to segment the pages of music into images of the individual measures of our dataset. We used the FastPAM k-medoids clustering algorithm with distances between measures based on Dynamic Time Warping to build clusters of measures that are rhythmically similar to each other. Using the ground truth, we evaluated the clusters we obtained. Although we notice that the selected method of clustering does not always separate the measures into clear rhythmic classes, we discuss possible solutions for it and suggest how the results can still be applicable in the crowdsourcing setting.

Keywords
Optical Music Recognition; Music score mapping; Staff detection; Barline detection; Measure detection; Dynamic Time Warping; k-medoids clustering; Rhythmic similarities

This page was intentionally left blank.

Preface

With this report, I conclude both my thesis project and my studies at the Delft University of Technology. It has been a very interesting, educational, and fun seven-and-a-half years of my life that I had the privilege of spending here and I am very glad I could finish with a project that aligns so closely with one of my biggest passions: music. During this period, I have not only had the opportunity to learn and study, but also to develop other skills and interests, such as serving as a board member at the TU Delft student orchestra Krashna Musika and building a tech startup, all of which I am very grateful for.

None of this could have happened without the people around me, so I would like to specifically thank a few of them. First of all, I would like to thank Cynthia Liem, who has been my supervisor during this project. She has pushed me in the right direction when necessary and has been very patient with me during this project which took way too long. Furthermore, I would like to thank Christoph Lofi for agreeing to be a part of my thesis committee and critically reviewing this work.

A special thanks goes out to Maarten and Mitchell, with whom I have shared almost every part of my time here in Delft. We have gone through almost every part of our studies together and I am convinced we are an invincible trio. I would also like to thank Ruben and Roy for forcing me to expand my interests and inspiring me the past couple of years. I am looking very much forward to what the future has in store for the five of us.

Finally, I would also like to thank my family. Thank you to my parents for supporting me through all of this, and thank you to my wonderful girlfriend for always being there for me.

Mathias Meuleman
Delft, December 2021

This page was intentionally left blank.

Contents

List of Figures

This page was intentionally left blank.

List of Tables

This page was intentionally left blank.

1
Introduction

Music has always been communicated in two ways: aural transmission, where music is played or performed and people can listen to it, and written transmission, where music is formalized in a document. These written formats come in many forms, one of which is western musical notation. These documents, mostly referred to as (music) scores, used to be handwritten, but were later printed on paper and can nowadays be found in digital format on computers, mostly as images or scans. Quite some effort has been taken to collect all of these digital scores and store them in central places or make them available to the public where possible. An example is the IMSLP library [16], where vast amounts of scores for classical music in the public domain are available. Collections such as these give rise to many opportunities, such as affordable scores and sheet music for musicians, or easy access to musical data for researchers. The formats in which these scores are stored, however, do have their limitations. When text is contained in PDF documents, the text is in many cases generated by the computer or can be transcribed as text through Optical Character Recognition (OCR), meaning it can be searched, edited, and analysed. For music, this is not the case. Scanned music scores are simply stored as images in PDF documents, meaning that none of the applications that are available for text can be applied to these scores. This means that a lot of applications that the digital age provides cannot yet be applied to music scores, and as such, there is a huge gap for potential left open. Parallel to Optical Character Recognition, research into Optical Music Recognition (OMR) is advancing, but it often does not perform well enough to unlock the possible applications of transcribed music.

In this first chapter, we will start by giving a short introduction to the field of Optical Music Recognition in Section 1.1. This will allow us to motivate our research and introduce the main idea behind this thesis in Section 1.2. Following that, we will introduce the concepts of music notation that are necessary for this thesis in Section 1.3. Next, we give a brief overview of the steps taken in the Optical Music Recognition field in Section 1.4 and explain how TROMPA, an international research project in this field [46], provided the main goals of this research in Section 1.5. After that, we come back to the problem statement in full and determine the scope of this work in Section 1.6, followed by the questions we want to answer in this thesis and the outline of this document in Section 1.7.

1.1. Introduction to Optical Music Recognition

By digitizing the contents of a music score, the information contained in it can be processed by a computer. This opens up a large number of possibilities for various target audiences. First of all, the field of musicology can benefit greatly from this, since digital scores are often not available. Cultural heritage can be safeguarded through digital scores since large amounts of historical significant collections of music are currently only available on paper. Musicians can be equipped with a “digital music stand”, on which the music score can be easily adapted to account for readability issues, or error corrections and annotations can be easily made and shared with other musicians. Furthermore, there is the obvious advantage of compiling large databases of digital information, creating possibilities for search and browsing, which are widely available for textual data, and further applications such as melody retrieval and similarity studies, just to name a few.

The digitizing process, or transcribing, is still relatively often done manually by experts. Several music notation applications can be found nowadays, which make their workflow more efficient. Examples of these applications are Sibelius[1], MuseScore[2], and Finale[3], in which users can input music either by keyboard and mouse or by using a MIDI keyboard. The large drawback of these systems is that inputting music is time-consuming and takes experience, and is therefore expensive. Automatic transcription of music scores could therefore dramatically improve this process, and this is what the field of Optical Music Recognition (OMR) is researching.

Automating transcription is unfortunately not an easy task. There are a lot of parallels to be drawn with the field of Optical Character Recognition (OCR), which focuses on transcribing written text. However, where most OCR tasks are considered to be solved, this does not hold for OMR. There are several reasons for this. First of all, OCR is generally a one-dimensional problem, where text is vertically evenly spaced out, but OMR is a two-dimensional problem, where duration and pitch make up the horizontal and vertical axes of sheet music, respectively [1, 6]. This can result in symbols influencing other symbols that are far apart on the page, such as a clef at the start of a system, which indicates what frequency the written notes entail, or accidentals that span a measure or an entire system width, see Section 1.3 for an overview of these symbols. Additionally, different symbols can be stacked on top of one another to signify multi-tones or chords, or they can be connected horizontally to one another. Stacked or connected notes can also be translated both horizontally and vertically relative to each other, meaning the connection can become stretched or tilted, see the work by Bainbridge et al. [1] for some examples. Besides, these translations and connections can often be done in different ways, or they can be purposefully left out. Decisions on if and how these transformations are used depend often on preferences of the musician, or larger contextual requirements, such as horizontal spacing to accommodate space for measures that are played at the same time. Furthermore, some symbols are relatively small in comparison to the notes, but have significant meaning, such as dots, dashes, and accents. There is also a large imbalance in the frequency in which symbols occur, increasing the likelihood of false positives when classifying these symbols [9]. The collection of musical symbols is also not entirely fixed, and new symbols are added regularly [14] and on top of all this, scanned music notation that is subject to OMR is often blurry, noisy, or fragmented. All of these difficulties make it impossible for simple digital systems to accurately transcribe music scores.

1.2. Motivation

Current OMR systems vary in the quality of their results, but a lot of the aforementioned possible applications require (near-)perfect results. For example, an orchestra that is playing a piece of music will need an accurate transcription of every symbol of the music, and using inaccurate results when compiling databases of music will make it easy for errors to influence results when this data is used and possibly accumulate when new data is generated from it. Since the current OMR solutions are still far away from this level of quality, some systems are being developed that focus on correction of existing OMR solutions instead. These systems require human input, and with the vast amount of music available for transcription, transcribing it all will cost a lot of time and energy. Being able to make this process as efficient as possible is therefore important, and we want to investigate if we can make progress in this aspect of OMR. More specifically, we want to see if we can more intelligently leverage the fact that music can be repetitive, and use that to reduce the time and energy required by human transcribers and correctors.

1.3. Introduction to Music Notation

Music notation is the manner in which music is communicated in written form. The de facto standard for this since the seventeenth century has been Common Western Music Notation (CWMN), of which the general principles are shared by almost all music that is written since then. CWMN consists of one or more music parts, which are depicted from left to right in staffs, synchronized in time. Staffs consist of one or more stafflines, the most commonly used staff in CWMN has five stafflines. The music is divided up into measures, which provide logical divisions and keep track of the current position in the music while playing. These measures are separated by barlines. Since measures are used to track the position of music in time, the different parts all have the same number of measures, synchronized in time. To make sure these parts fit on pages, they are broken up at barlines and placed underneath each other on the page, or on the next page if there is no more space on the page left. This pattern repeats until the end of the piece. Each such section that spans a page's width is called a system, which consists of a set of staffs, all divided into synchronized measures. We

[1]https://www.avid.com/sibelius
[2]https://musescore.org/
[3]https://www.finalemusic.com/

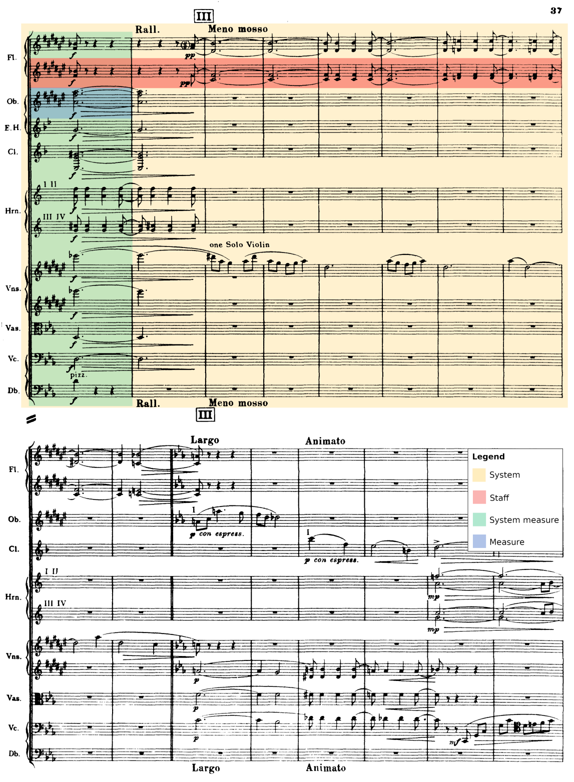
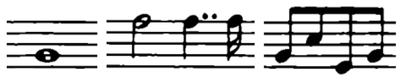
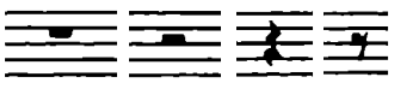


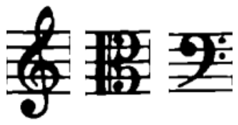
Figure 1.1: Example structure of a page of music.



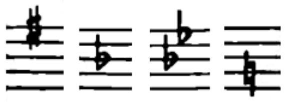
(a) Examples of notes with varying duration.



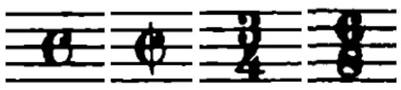
(b) Examples of rests with varying duration.



(c) Examples of the three most common clefs.



(d) Examples of sharps, flats, and naturals. Together they are called “accidentals”.



(e) Examples of varying time signatures.

Figure 1.2: Examples of frequently occurring music symbols.

call the collection of measures that overlap at a given point in time a system measure, to differentiate them from the individual measures. It can be customary to leave staffs out of a system if the parts in it have nothing to play for the duration of that system, causing differences in the number of staffs between systems. In Figure 1.1, examples of all these elements of CWMN are shown. The first of the two systems is highlighted in yellow, the second staff of the first system is highlighted in red, the first system measure is highlighted in green and the first measure of the third staff is highlighted in blue. If we count the staffs present in both systems, we also see that the first system has one staff more than the second system, because an additional part is left out of that system to save space on the page.

Going to a smaller level, we touch upon the contents of the individual measures. A large collection of symbols are used to convey musical information, we introduce the ones that are important for this thesis in Figure 1.2. There we see notes and rests, which are the two most common symbols, as well as the three most commonly used clefs, and sharps, flats, and naturals, which together are called “accidentals”. The current applicable clef and accidentals are often shown in the first system measure of each system, as is also the case in Figure 1.1. The vertical placement of notes indicates their pitch, together with the clefs and accidentals that might occur in their system or their measure. Their horizontal relation to each other, as well as the type of note, flags, and rests, indicate the duration of the note. The combination of durations is what we call a rhythm. Time signatures determine what total duration each measure has. Each time signature consists of a numerator and a denominator. The numerator indicates how many beats each measure is given, and the denominator indicates which note is defined as one beat. If the time signature is e.g. three-four time, which is the third example of Figure 1.2e, each measure has three beats, where a quarter note is one beat. The first two examples in Figure 1.2e indicate four-four time and two-two time, respectively.

One special case that is worth noting is the empty measure, which consists entirely of a single, full-measure rest without any notes. This indicates that in this measure the instrument has no role to play, as can be seen in e.g. measures 3 through 8 of staffs 3, 4, and 5 of the first system in Figure 1.1. When a staff consists of only empty measures, we call that staff an empty staff. The empty staff is the only type of staff that can possibly be left out from a system to save space.

1.4. A Brief History of OMR

The OMR field has been active for over 50 years now, starting with Pruslin in 1966 [31]. Since then, several different approaches have been tried, which ultimately led to a generally agreed upon pipeline in which the OMR task could be subdivided into smaller subtasks. This pipeline was first coined by Bainbridge and Bell [1] and was later elaborated on by Rebelo et al. [33] in a vast literature review. Both papers indicate that the majority of research up to that point could be segmented into the various stages of this pipeline, and in fact, a lot of research from that point on falls into place as well when considering this framework. Although research following this pipeline progresses, most of it suffers from a lack of fixed evaluation metrics. Devising suitable error metrics in OMR research is a hard problem, and shared datasets are hard to come by, making it difficult to compare different solutions [3, 6].

There have been deviations though, since not all stages are equally easily solvable. With the increasing research and application of deep learning, attempts at full-pipeline and end-to-end OMR have started to occur as well [7, 30, 48]. The preliminary results look promising, but the research effort is scoped very tightly, making it so that a lot of work needs to be done before these approaches can be considered generally applicable. This is also hampered by the main struggle of deep learning: the lack of sufficient data. Some datasets have been compiled, but often the tradeoff has to be made between quantity and representation of different fonts, typesets, or handwriting styles. Rebelo et al. [32] have scanned, processed, and annotated real-world scores, but this process is very costly and so the resulting dataset is quite small in comparison to the vast amount of music out there. In contrast, Tuggener et al. [47] have used synthetic data, allowing them to automatically annotate the scores and therefore dramatically increase the size of the dataset. However, this means that many real-world typesets and fonts are not considered, even though they render the music in a few different fonts, and therefore this dataset probably is less representative of real-world scores.

Due to all of these problems, current workflows tend to pipe the result of an OMR system into a music notation application for human error correction. This process of human correction at the end is still very expensive. Bellini et al. [3] claim that even a recognition rate of 90% does not make an OMR system attractive to music copyists. Attempts at human-aided OMR are therefore made, where the focus lies on human feedback in several stages of the pipeline instead of at the end of the process. The hope is that this will either reduce the total time required of expert users or make non-expert user feedback an option to improve the performance of such an OMR system in comparison to fully automated versions. Examples of such research are Gamera, an ease-to-use scripting environment for OMR, developed by MacMillan et al. [22]; Ceres, an adapted OMR system that takes human feedback into account, developed by Chen et al. [8, 9]; and Allegro, which takes a more user-centred approach and provides an easy framework for users to transcribe the contents of simple handwritten scores, developed by Burghardt and Spanner [5].

1.5. The TROMPA Project

Towards Richer Online Music Public-domain Archives (TROMPA) is an international research project in which scientists, scholars, and musicians cooperate to push towards the goal of making more applications available to the music domain and increase engagement with classical music and music scores [46]. One of their main research themes is called “Scanned score analysis” and focuses on improving OMR and bringing together different techniques to make large-scale transcription of music scores feasible, so that those transcriptions can be used in a variety of other research themes. Within this research theme, work has been done on human-aided OMR through crowdsourcing, in which a large userbase is asked to interact with the results of an OMR system to check and correct the results of the system.

One of the issues that arises when dealing with large music works like this is the repetitiveness of correcting and checking measures of the music score. Especially in large ensemble works, it occurs frequently that several instruments or instrument groups don’t play continuously, but instead have parts of the piece where there is a gap in their score. Since these instrument groups still need to be aligned with the rest of the ensemble in the music score, these parts are indicated by empty measures. Over the different instrument groups, the total of these empty measures can add up quickly, which raises the question of whether this characteristic can be utilized to improve the workflow of this human-aided OMR system, and perhaps reduce the work that has to be performed by users.

Since our work is inspired originally by the TROMPA project and its research, we will be focussing on this crowdsourcing setting as the main use case for our research. We do however recognize that this is not the only possible use case, and that our research could, and hopefully will, be useful in other applications of OMR.

1.6. Problem Statement

This leads to the main goal of this thesis. We want to investigate whether the work needed for this OMR task can be made easier by finding similarities in the music scores on the measure level. If it is indeed possible to find such similarities, and we can separate these similar measures with reasonable accuracy from the remainder of the score, we can share collected data across similar measures, whether it is acquired through automated transcription or human input. This also means that the benefits of such a system would not be limited to the TROMPA project, but could be applied in a wider OMR context. We will study these similarities on PDF documents that contain the scores, since those documents are the main targets for (automatic) transcription in the TROMPA project. The scope of the studied music documents is defined as follows:

- We focus on music written for ensembles of musicians. These ensembles can range from small chamber orchestras of 8 people to full-sized symphony orchestras. We do not include music for smaller ensembles, since similarities in music, such as empty measures, are more likely to occur in larger ensembles.
- We focus on music scores that are available in typeset fonts, since the vast majority of these ensemble works can be found as typeset in public domain collections, such as the Petrucci Music Library (IMSLP) [16].

To be able to find these similarities, we first need to be able to accurately find the measures in a given score. Therefore, a preliminary step needs to be taken which can segment a music score in PDF format into its individual measures, to which we dedicate the first part of our research. After that, we can use the extracted measures and focus on the similarity study. The similarity study will focus on the rhythmic contents of the measures. Out of the two main axes of written music, duration and pitch, changes in duration and rhythm will have a more profound impact on the visual representation of the measures, as notes or rests are added or removed with changes in rhythm. These changes should therefore be most notable and we should therefore be able to more easily compare the measures on their rhythmic component.

1.7. Research Questions and Thesis Outline

In this thesis, we want to investigate whether we can find similarities between the measures of a music score. We have set the scope of the music we want to focus on and we have identified the steps we need to take. With this in mind, we define the following main research question:

> ***RQ***: *How can we design and evaluate a digital system that finds similarities in scanned music scores on the measure level?*

To structure this further, we provide the following subquestions:

> ***SQ1***: *What data are needed to design and evaluate a digital system which can find similarities and how can we prepare the data for this task?*
>
> ***SQ2***: *How can we reliably detect the measures in a scanned music score?*
>
> ***SQ3***: *How can we find similarities between detected measures based on rhythmic information?*

Given this research question and its subquestions, we structure this thesis as follows: In Chapter 2, we provide a theoretical background for all the material used in this thesis. SQ1 is answered in Chapter 3, where we discuss the requirements and acquisition of the data and how this data is prepared for use in our system. Chapters 4 and 5 answer SQ2: in Chapter 4 we discuss staff detection and in Chapter 5 we discuss barline detection, the two components needed for measure detection. Following that, we show how clustering of these detected measures can be performed in Chapter 6 to answer SQ3. Finally, we discuss the results in Chapter 7 and conclude and provide recommendations for future work in Chapter 8. The general approach from data acquisition to measure clustering is schematically visualized in Figure 1.3.

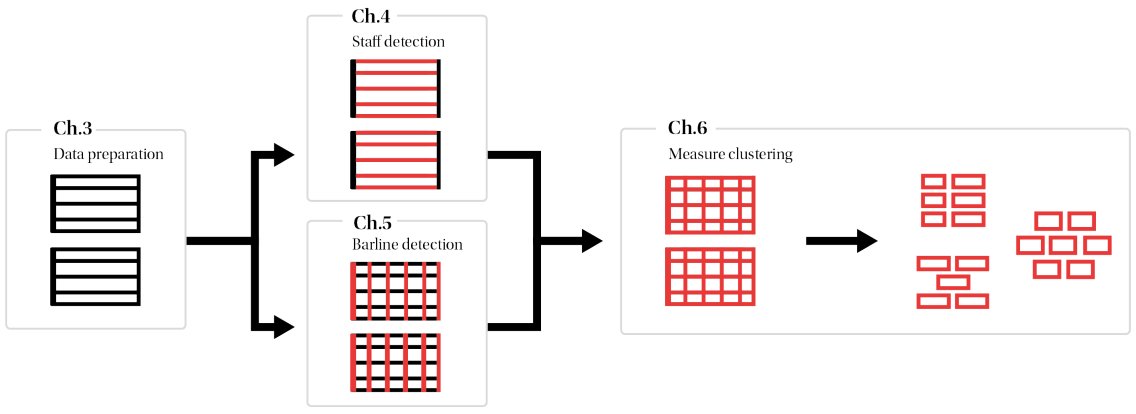


Figure 1.3: Schematic outline of this thesis.

1.8. Terminology

We recognize there is a clash in terminology between the music domain and the computer science domain. To avoid ambiguities throughout this thesis, we want to clarify our usage of some relevant terminology. Both the terms “system” and “measure” have meanings in both domains, and we choose to use them in the music domain, as introduced in Section 1.3, instead of the computer science domain. We will use the term “digital system” instead of “system” in the computer science domain to differentiate it from the systems in music, as we have done in our main research question. Furthermore, we will use the term “metric” when a measure in the computer science domain is meant, although we realise that this is semantically incorrect. Metrics adhere to the triangle inequality property, meaning that for any three points x, y, z the distance from x to z $d(x,z)$ can never be larger than the sum $d(x,y)+d(y,z)$. Measures on the other hand do not adhere to the triangle property, but we choose to use “metric” incorrectly to avoid confusion in terminology between the two domains. We will clarify whether the triangle inequality holds when necessary.

2 Related Work

We use this chapter to introduce background from the literature. In Section 2.1, we introduce various methods of staff detection, and more thoroughly discuss one of these algorithms. Section 2.2 is used to introduce the research into measure detection algorithms. We introduce Dynamic Time Warping in Section 2.3, after which we use Section 2.4 to introduce the main concepts of clustering techniques. Finally, we provide a short introduction to symbolic melodic similarities in Section 2.5.

2.1. Staff Detection Algorithms

One of the main steps in the OMR pipeline that is extensively described by Rebelo et al. is staffline detection and removal [33]. This is generally performed after some image preprocessing steps and helps with the detection and classification of musical symbols. Most staffline detection algorithms use the *staffline-height* and *staffspace-height* reference lengths that are determined in the preprocessing step. These lengths represent the staffline thickness and the vertical distance between two consecutive stafflines within a staff. As Fujinaga showed in his dissertation [14], these reference lengths can reliably be found through run-length encoding, where each run of equal values in a binary image is represented as a single digit. The most common black run-length is used as the *staffline-height*, the most common white run-length is used as the *staffspace-height*.

Different staffline detection methods have been proposed previously. Fujinaga used a method of horizontal projections [14], as can be seen in Figure 2.1 which was taken from Bainbridge et al. [1]. Adaptations to this approach are for example made by Bellini et al., who describe a filtering technique on the peaks found through the horizontal projection [2]. Bainbridge and Bell suggest applying these projections to smaller sections of a page, rather than the whole page, and logically connecting the found staffline segments, which makes staffline detection more robust against small rotation and skew in the scanned page [1]. Szwoch uses horizontal projections of small vertical slices of a page [44]. These projections are then processed to obtain candidate points on vertical lines which are horizontally connected if their vertical distance is small enough. Miyao uses vertical scan lines to extract candidate points for stafflines [24]. On each vertical scan line, if a black run is close to the length of a staff width, its middle point is extracted as a candidate point. These points are then horizontally connected using a dynamic programming approach.

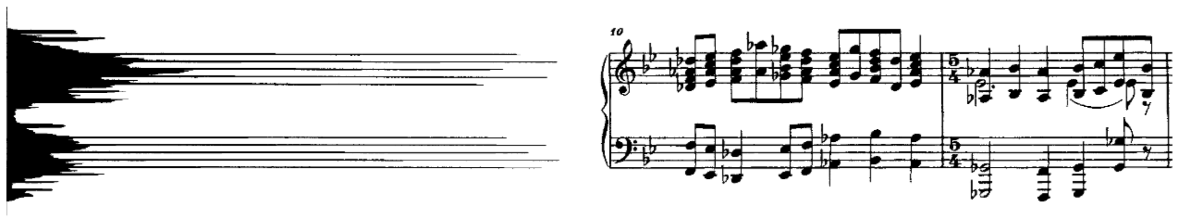


Figure 2.1: An example of horizontal projection of a small excerpt of music. The stafflines create clear peaks in the histogram. These are figures 11 and 10 from Bainbridge et al. [1], respectively.

2.1.1. Staff Detection According to Dalitz et al.

Dalitz et al. have done a comparative study of staff removal algorithms in 2008 [10]. Staff removal is not straightforward, since stafflines often intersect musical symbols, so removing all staffline pixels would break up musical symbols, hampering object classification performance in a later stage. Most of the staff removal algorithms considered by Dalitz et al. follow a two-step approach: first find the positions of the stafflines using a staff detection algorithm, and then use a given metric to determine which of those pixels to remove and which to keep intact. For the first step, a new staffline detector was created. Dalitz et al. make the case that the candidate points which are used by Miyao and Szwoch are only capable of polygonal approximation, given that they are bounded by a set of vertical lines. The stafflines in the scanned music are often not entirely straight, and such an approximation is therefore not accurate enough for the staff removal algorithms under consideration. The authors introduce a skeletonization step as a generalization of Miyao's algorithm, which yields a staffline skeleton of *staffsegments*. Then they explain which steps are taken to find the staffs based on the *staffsegments*. Although Dalitz et al. use this staff detection for further comparison of staff removal algorithms, we will use it to develop our own staff detection algorithm in Chapter 4.

Skeletonization

The skeletonization step first does a windowed extraction of horizontal black runs, with a window of size $1 \times$ `window_size` and a threshold on the blackness of the window. If the average blackness of the pixels in the window exceeds the blackness threshold, that pixel is selected as part of a *staffsegment*. To make sure black runs are extracted in full, black pixels at the start and end of a detected black run need to be "rescued", since they are close to the transition from black to white. This causes their average blackness to fall below the set threshold, although they are part of the black runs that need to be detected. To make sure these are included, any black pixels that are directly adjacent to the selected *staffsegment* pixels also included in the selection. These horizontal black runs are then thinned vertically by replacing each vertical column of black pixels by its middle pixel only, yielding horizontal black runs that are one pixel in height. This procedure is described in Algorithm 1. The SKELETONIZATION function takes a binary input image of size $m \times n$, the `window_size` w, and the threshold value t.

The VERTICAL_SEGMENT and MIDDLE functions are not included for brevity. The VERTICAL_SEGMENT function yields t and b, the y-value of the top and bottom coordinates of a vertical black segment given a pixel position (i, j). It also checks whether there is a vertical segment to be found in the column next to column i, and returns the black pixel that is closest to a in column i as r. When there is no such pixel, $r = 0$ is returned, indicating the end of the current black run. The MIDDLE function determines the middle point of the current column i, given the found vertical segment and the previous middle point a. This middle point is the pixel that is selected as the thinned result for this column.

The resulting segments are filtered by length, where all segments shorter than twice the *staffspace-height* are removed. The remaining segments are now considered as the *staffsegments* to be used in the staffline detection algorithm.

The Staff Detection Algorithm

Based on the method described by the authors and the extensive documentation that can be found in the publicly accessible implementation[1], the staff detection algorithm works as follows:

1. Take as input the original image and an optional parameter `numlines`, which indicates the number of stafflines per staff.

2. Obtain the *staffsegments* by extracting the horizontal runs, thinning the runs vertically and removing the ones that are too short.

3. Add vertical links between *staffsegments* that overlap horizontally and have a vertical distance of approximately *staffspace-height* + *staffline-height*.

4. Remove *staffsegments* that have no vertical links. Each group of connected *staffsegments* is now considered a staff. The staffs are labelled, and each *staffsegment* in that staff is labelled based on the number of vertical link steps that precede it. Now each staffline consists of several *staffsegments*.

5. Within each staff, combine the *staffsegments* that overlap to make sure no *staffsegments* in each staffline overlap.

[1] `http://music-staves.sourceforge.net`

Algorithm 1 Skeletonization

```
function BLACK_RUNS($I \in \{0,1\}_{m\times n}$, $w$, $t$)
    $O \leftarrow 0_{m,n}$
    $f \leftarrow 1$
    for each $i \leftarrow 1..m, j \leftarrow 1..n$ do
        if $\frac{1}{2w}\Sigma_{k=i-w}^{i+w} I_{k,j} \geq t$ then
            $O_{i,j} \leftarrow 1$
            if $f = 1$ then
                while previous column $p$ has $I_{p,j} = 1$ do        ▷ Recue black pixels at start of run
                    $O_{p,j} \leftarrow 1$
                $f \leftarrow 0$
        else if $f = 0$ then
            if $I_{i,j} = 1$ then
                $O_{i,j} \leftarrow 1$
            else
                $f \leftarrow 1$
    return $O$

function THINNING($I \in \{0,1\}_{m\times n}$)
    $O \leftarrow 0_{m,n}$
    for each $i \leftarrow 1..m, j \leftarrow 1..n$ if $I_{i,j} = 1$ do
        $a \leftarrow 0$
        repeat
            $t, b, r \leftarrow$ VERTICAL_SEGMENT($i, j, a$)
            $a \leftarrow$ MIDDLE($t, b, a$)
            $O_{a,j} = 1$
            for $k \leftarrow t..b$ do        ▷ Do not consider these pixels anymore
                $I_{k,j} = 0$
        until $r = 0$        ▷ Terminate when no next segment is found in this run
    return $O$

function SKELETONIZATION($I \in \{0,1\}_{m\times n}$, $w$, $t$)
    return THINNING(BLACK_RUNS($I, w, t$))
```

6. Remove staffs that have less than `numlines` stafflines. If `numlines` was not provided as an input, staffs that have less than the most frequent number of stafflines are discarded.

7. If a staff has more stafflines than `numlines`, keep removing the shortest of the upper and lower stafflines until the number of stafflines equals `numlines`. If `numlines` was not set, all stafflines that are much shorter than the longest staffline are removed instead.

8. Remove staffs that horizontally overlap with wider staffs. If any of the remaining staffs are sufficiently close together horizontally, merge them.

9. If any stafflines are broken, interpolate between the *staffsegments* to join them together.

This algorithm yields a collection of staffs, each of which has a collection of stafflines. These are used by the various staff removal algorithms which determine which parts of the stafflines should be removed to keep the musical symbols intact as much as possible. These are evaluated by Dalitz et al. on a set of 32 pages of music with varying numbers of stafflines per staff. For each page, both the original as well as a version without stafflines were generated. The difference between these two images serves as the ground truth for staff removal. Next, various deformations were applied to the images before the staff removal algorithms were tested on them.

2.2. Measure Detection Algorithms

There is very little research done into measure detection algorithms as a stand-alone topic. In the OMR pipeline, measures are not regarded as objects of interest. Instead, barlines are often included in the object classification stage, after which the detected music can be logically divided into measures using the spatial relationship between the barlines and other elements. There are two publications that have proposed measure detection algorithms.

Vigliensoni et al. have developed a measure detection algorithm that uses *staff grouping hints* [50]. After preprocessing, a staff detection algorithm is run, which is the same algorithm that Dalitz et al. presented, with a fallback on Miyao's staff detection algorithm. The staffs are then removed using the method presented by Roach and Tatem [34]. Finally, barline candidates are isolated and filtered to find the final barlines. With the staffs and barlines detected, the measures can be constructed. Crucially, this algorithm uses a *staff grouping hint*, which is a manual annotation that needs to be created for each page, indicating how many systems are present on the page and how many staffs are present within a system. This hint is used to determine whether the Dalitz staff detection algorithm yields satisfactory results, and if not, fall back onto Miyao's algorithm.

The second publication that focuses on measure detection, is a deep learning approach by Waloschek et al. [51]. A large collection of pages was manually annotated with measure bounding boxes. A neural network was used to train a model that can detect those measures. The use case for this research did not require very precise annotations, so they only roughly outline the measure bounding box, meaning measures can often overlap slightly with adjacent measures.

An important observation to make for both these measure detection algorithms is that they work only on the system measure level and not on the measure level.

2.3. Dynamic Time Warping

Finding the distance between two time series is an often performed task. The goal is to give time series that have similar events occurring a small distance, even though these events can be offset across the time axis. This makes it so that a simple metric such as pairwise Euclidean distance is not an adequate metric because small offsets in the time direction would quickly increase the distance between time series that are quite similar. As an alternative, a dynamic programming approach was proposed by Bellman and Kalaba [4], which was applied to time series by Kruskal and Liberman [20] and has been known as Dynamic Time Warping (DTW) since. We give a brief overview of its workings, and show how it can be optimized.

Given two time series X of length m and Y of length n with:

$$X = x_1, x_2, \dots, x_m \tag{2.1}$$

$$Y = y_1, y_2, \dots, y_n \tag{2.2}$$

a cost matrix c of $m \times n$ is constructed where element (i, j) represents the total cost of warping Y to X up to element i and j of X and Y, respectively. We construct the cost matrix through the following recurrence relationship:

$$c(i,j) = d(i,j) + \min\{c(i-1,j-1), c(i-1,j), c(i,j-1)\}, \text{ with } c(1,1) = d(1,1) \tag{2.3}$$

The two time series are warped to match each other according to the optimal warp path W with length K through the cost matrix, with:

$$W = w_1, w_2, \dots, w_k, \dots, w_K \tag{2.4}$$

The cost of each element (i, j) in c is calculated as the minimal cost yielded from any allowed operation for each next element in the time series, starting at the first element for both series. The allowed actions are matching, insertion and deletion, which are enforced by the following constraints:

- Boundary conditions: $w_1 = (1,1)$ and $w_K = (m,n)$, meaning the warp path has to start and end at the extremes of the main diagonal of the cost matrix.
- Continuity and monotonicity: Given $w_k = (a,b)$, it must hold for the next step $w_{k+1} = (a', b')$ that $a \leq a' \leq a+1$ and $b \leq b' \leq b+1$, meaning that steps in the cost matrix are limited to adjacent cells only, with the diagonals included, and that the steps are always going "forward" in the cost matrix.

The distance between X and Y according to the optimal warp path is the DTW distance: $DTW(X,Y) = c(m,n)$

Allowing insertion and deletion as two of the allowed operations gives DTW two major advantages over Euclidean distance. The matching between the two time series is essentially warped to match them as closely as possible, meaning the distance measure reflects possible shifts or offsets in time more naturally. Besides, this also allows time series of different lengths to be compared, something that pairwise Euclidean distance does not.

2.3.1. Improving DTW

The downside of the DTW algorithm is that it has a quadratic time and space complexity, which is easy to see. Each cell in the cost matrix is computed in constant time and the number of cells scales quadratically with the input size. A lot of research effort has been put into methods of improving the time and space complexity of DTW, two of which we will highlight here. First, the possible warp paths through the cost matrix can be reduced by adding a window constraint to the cost matrix. The Sakoe-Shiba band [38] and Itakura parallelogram [17] do this by restricting the warp path options to completely lie within either a band of set radius across the diagonal or a parallelogram with two vertices on $c(1,1)$ and $c(m,n)$. The cells outside these windows are not considered anymore for the warp path, speeding up the DTW distance calculation. Second, the DTW algorithm can be sped up by data abstraction. By reducing each time series to a smaller size, while keeping the information within it preserved as well as possible, the size of the cost matrix can be reduced by a given factor, reducing the necessary calculations and speeding up the algorithm as well.

The FastDTW algorithm, proposed by Salvador and Chan [40], makes use of both these concepts to construct an adaption to DTW that approximates the true DTW warp path in linear space and time. It uses a multilevel approach with respect to data abstraction in three key steps: coarsening, projection and refinement. During the coarsening step, the time series are shrunk into increasingly lower resolutions, abstracting the data further every step. A warp path at the lowest resolution is calculated, and the result is projected onto the next higher resolution, determining which cells of the cost matrix the warp path will pass through given the initial warp path. During refinement, the optimal path in the neighbourhood of the projected path is found, refining the projected solution. The two steps of projection and refinement are repeated for each next resolution until a warp path for the original time series is found.

2.4. Clustering

Clustering attempts to group objects in such a way that similar objects are put in the same cluster, and dissimilar objects are assigned to separate clusters. This technique has been used in various data exploration and classification tasks in the past decades. Two common clustering techniques are k-means [23] and k-medoids [18] clustering. Both create k clusters, where k needs to be defined beforehand, but where k-means uses spatial points to define a cluster center, k-medoids uses datapoints only. This makes k-medoids clustering useful when datapoints cannot be modelled as points in space, but when (dis)similarities between datapoints are available. Non-Euclidean distance measures, such as Dynamic Time Warping, have been successfully used in combination with k-medoids [28], and can be more effective [29] in k-medoids than in k-means.

The traditional algorithm for computing the k-medoids clusters is the Partitioning Around Medoids (PAM) algorithm [18]. This algorithm takes as input a distance matrix, indicating the distance between all pairs of datapoints, and k the number of clusters, and then attempts to minimize the total deviation TD

$$TD = \sum_{i=1}^{k} \sum_{x_c \in C_i} d(x_c, m_i) \tag{2.5}$$

which is the sum of all distances between each point $x_c \in C_i$ and its assigned medoid m_i. Minimization happens in two phases. The first phase is called BUILD, and it iteratively initializes k cluster centers. The first cluster center is chosen as the point which has the smallest sum of distances to all other points. After that, each next cluster center is chosen as the point that decreases TD the most, until k medoids are selected in the set $M = \{m_1, \dots, m_k\}$. Each point is then assigned to the medoid to which it has the smallest distance. Next, the SWAP phase tries to improve the selection of cluster centers. For each pair $(x_c, m_i), x_c \notin M$ the decrease in TD is measured when m_i would be swapped out with x_c as new medoid. The pair with the largest decrease in TD is then swapped. This is repeated until no further swap decreases TD, at which point the algorithm returns a local optimum as the result.

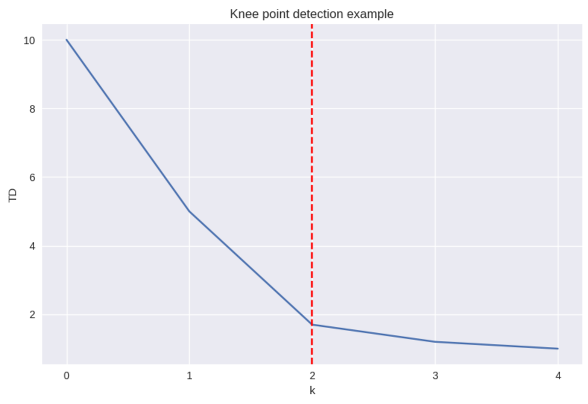


Figure 2.2: An example of visual knee point detection. The knee point at $k = 2$ is highlighted.

The original PAM algorithm has a runtime of $O(n^2 k)$ for the BUILD phase, and $O(k(n-k)^2)$ for each iteration of the SWAP phase when implemented in an efficient way [42]. This makes it a very expensive algorithm, so various optimizations and alternatives have been proposed. CLARA (Cluster LARge Applications) [19] runs the PAM algorithm on a subset of the original data and reduces the runtime this way. Once a clustering is found, the remaining datapoints are assigned to the closest cluster and *TD* can be computed. This is repeated multiple times, the clustering with the smallest *TD* is returned. CLARANS (Clustering Large Applications based on RANdomized Search) [27] is another adaptation to PAM, which models the clustering search space as a graph. Each node of the graph is a possible selection of medoids from the input data, nodes are connected when their selected medoids only differ by one, so when they are one swap away. Using this model, PAM is modelled as finding the minimum *TD* by searching through the graph, and CLARA corresponds to finding the minimum of *TD* on a subgraph, and therefore can possibly miss out on the optimal solution by excluding it from the start. CLARANS employs randomized search on the total graph, doing a greedy search where each first edge that reduces *TD* is followed until no edge does this. This search is initialised randomly and repeated multiple times, the node found with the lowest *TD* is chosen as the resulting clustering.

In 2019, Schubert and Roosseuw introduced FastPAM and FasterPAM [42]. FastPAM gives an optimization for the SWAP phase, reducing its runtime dramatically by exploiting redundancies in its computations, while guaranteeing to find the same results as PAM. The runtime of the FastPAM SWAP phase is reduced to $O((n-k)n)$, reducing the runtime by $O(k)$. FasterPAM even goes a step further by greedily performing swaps in a similar way that CLARANS does. This does reduce the runtime even further, but has the disadvantage of yielding only approximations to the original PAM algorithm. For both FastPAM and FasterPAM, the bottleneck of the algorithm is no longer the SWAP phase, but the BUILD phase. Since the SWAP phase has become much more efficient, however, the quality of the medoids selected in BUILD is now of less importance, since more SWAP iterations can be performed to fix the quality. This means that using random initialization is now an interesting alternative to the BUILD phase, and indeed the researchers show in their experimental evaluation that uniform random initialization can be a good choice. With this replaced BUILD step, the runtimes of FastPAM and FasterPAM are reduced even further.

2.4.1. Selection of k

All clustering algorithms we discussed need to have the number of clusters k predefined. Therefore we need a way to find a good value of k. One commonly used heuristic is the elbow method, or knee point detection. This attempts to find the cut-off point where increasing the number of clusters k no longer yields large enough improvements. This “knee” is visualized in the k vs *TD* plot in Figure 2.2. In this artificial example, it is quite easy to identify the “knee” at $k = 2$, but this is not always so straightforward. More smoothed out graphs or higher dimensional data can make it non-trivial to manually detect the optimal k. Therefore, some automatic methods have been proposed. An overview of some of them can be found in a review by Salvador et al. [39]. Satopää et al. [41] have proposed the “kneedle” algorithm for finding knee points in a collection of discrete data points. This method tries to find the maximum (or minimum, depending on orientation) of the graph when the graph is rotated in such a way that the line defined by the points $(x_{\min}, y_{\min})$ and $(x_{\max}, y_{\max})$ is parallel to the x-axis. When reversing the rotation of the found extreme, the “knee” or “elbow” is the x-value of the point on the original graph.

2.4.2. Cluster Evaluation

There are generally two categories of cluster performance evaluation. Internal evaluation tries to evaluate the cluster performance by means of a given function, which will try to quantify performance based on cluster form, tightness, and other intrinsic characteristics of clusters. Examples are the Davies-Bouldin index [11], the silhouette coefficient [37], as well as the k-selection method we described above. External evaluation, on the other hand, compares the clustering to an existing ground truth. This gives a better insight into the actual validity of the cluster results. One such external evaluation metric is the V-measure [36], which we will use later on. It is composed of two components: the homogeneity of the clusters and the completeness of the clusters. The homogeneity of a cluster h determines how many different classes are present in what quantity of a cluster, in the ideal case this is just one class. The completeness of a cluster c determines how many different clusters are needed in what quantity to represent a class. Mathematically they are defined as:

$$h = 1 - \frac{H(C|K)}{H(C)} \tag{2.6}$$

$$c = 1 - \frac{H(K|C)}{H(K)} \tag{2.7}$$

where H is the Shannon entropy function [43]. The V-measure is the harmonic mean between these two components, given by

$$V_\beta = \frac{(1+\beta) \times h \times c}{(\beta \times h) + c} \tag{2.8}$$

Generally β is set to 1, which means homogeneity and completeness are weighted equally in the V-measure computation. However, either one can be weighted heavier over the other by increasing or decreasing β. The definitions of homogeneity, completeness and V-measure show that they are all bounded between 0 and 1, where 1 is the most desirable outcome and 0 the least.

2.5. Symbolic Melodic Similarities

Research into symbolic melodic similarity is concerned with finding similarities between sequences of notes. The pitch, duration, and other relevant information for these notes are available and are leveraged to define similarity metrics in sequences. Especially the Symbolic Melodic Similarities competition of MIREX [12] has helped focus attention on this topic. The surveys by Müllensiefen et al. [26] and by Velardo et al. [49] on this research give a good overview of different approaches taken to obtain solutions to this problem. There is a large number of various directions to these approaches, since, according to [49], “melodic similarity spans several elements of music theory, ethnomusicology, cognitive science, and computer science, all of which have to be considered simultaneously”.

For the purposes of this work, we focus on research where rhythm and pitch are encoded separately, so that the rhythmic component can be easily isolated. Most methods of rhythmic encoding use the same concepts as dictated in the MIDI specification [45]. A good example is the work by Mongeau and Sankoff [25], who encode notes as (pitch, duration) tuples, where duration is specified in terms of sixteenth notes.

This page was intentionally left blank.

3 Data Description

n this thesis, we want to focus on music scores for larger ensembles. Music scores are generally available as images embedded in PDF documents, since most of them are scans of works previously published on paper. When these scores are digitized through transcription, they can be exported in either native proprietary formats, such as `.mscz` for MuseScore, `.mus` for Finale, and `.sib` for Sibelius, or as open standards such as MusicXML [15] and MEI [35]. We differentiate between image-based scores and scores in any of the digitized formats by referring to them as printed scores and symbolic scores, respectively. We present in Section 3.1 our data requirements and acquisition methods. Following that, we show how the symbolic data can be mapped to the printed scores to evaluate our experiments in a later stage in Section 3.2.

3.1. Requirements and Acquisition

Since there is a lot of variation in style, contents, and notation of classical works, we want to sample a broad selection of musical pieces. There are a few things to keep in mind when selecting data for this work. To test this work on real-world data, we use printed scores that are readily available from places where most musicians and researchers would find printed music scores. To make sure results can be verified, however, we need symbolic scores that contain the same musical information as their printed counterparts. We aim to find pieces suited for larger ensembles. Measure similarity in music, such as empty measures, occurs much more frequently in larger ensembles, so small ensemble pieces will generally not be suitable. Therefore, we will focus on works for larger ensembles, which we choose arbitrarily as ensembles with at least 8 individual parts. We want to consider pieces from the 18th, 19th, and 20th century, since works of those periods are notated in CWMN, have varying ensemble compositions, and are still regularly performed. Users of these scores could therefore benefit from a wide range of possible applications of digital transcriptions of these scores. Since symbolic scores are significantly harder to find than printed scores, we focus on that first, and attempt to find suitable printed scores of the same piece of music. Most of the printed scores that adhere to these requirements are typeset instead of handwritten, since they have been made available by publishers. Although we limit ourselves to these typeset sources, we want to make sure to include different typesets and fonts, which correspond to the varying styles of notation these ensembles generally work with.

Given these requirements, we started data acquisition by searching for symbolic scores. The sources we ended up using are the OpenScore project[1], Verovio[2], the Musedata collection by CCARH at Stanford[3], and MEI sources from one of the TROMPA partners. We chose the `music21` Python toolkit as a means to read and process this symbolic data, for its abilities to interpret many different symbolic formats. As a result of this choice, however, we omitted any LilyPond data, which uses a text-based syntax instead of an XML-like structure, since this is not supported by the toolkit and we have a sufficiently large sample without it. Other sources of symbolic data were discarded mainly because we could not easily verify the accuracy of the symbolic scores.

With these symbolic scores collected, the next step is to find matching printed scores. We obtained most of these through IMSLP. They represent a set of different fonts and typesets, but to be complete, we also

[1]`https://musescore.com/openscore-transcriptions`
[2]`https://www.verovio.org/musicxml.html`
[3]`https://wiki.ccarh.org/wiki/MuseData:_Ludwig_van_Beethoven`

Table 3.1: Overview of selected music scores.

Title	Composer	Source	#Pages	#Measures	#Empty Measures
Brandenburg Concerto No. 5	Bach, J.S.	Verovio	17	1432	207
Symphony No. 1	Beethoven, L. van	Musedata	44	10384	3275
Symphony No. 2	Beethoven, L. van	Musedata	61	11819	2929
Symphony No. 3	Beethoven, L. van	Musedata	89	18310	4694
Symphony No. 4	Beethoven, L. van	Musedata	38	5733	1209
Symphony No. 5	Beethoven, L. van	Musedata	87	17799	4748
Symphony No. 6	Beethoven, L. van	Musedata	79	12413	2893
Symphony No. 7	Beethoven, L. van	Musedata	89	18801	4724
Symphony No. 8	Beethoven, L. van	Musedata	133	16494	4708
Symphony No. 9	Beethoven, L. van	Musedata	189	34821	10000
Symphony No. 3	Brahms, J.	TROMPA	114	11640	4002
Symphony No. 5	Bruckner, A.	TROMPA	180	26326	7833
Symphony No. 9	Bruckner, A.	TROMPA	185	30538	15298
The Planets	Holst, G.	OpenScore	187	33695	16729
Symphony No. 4	Mahler, G.	TROMPA	128	17852	8350
Symphony No. 41	Mozart W.A.	OpenScore	56	9069	2154
Ouverture 1812	Tchaikovski, P.I.	OpenScore	88	6222	1073
Total			**1764**	**283348**	**94827**

included a few printed scores that were generated by the music notation software Finale, instead of scanned from published scores. These are the only scores that were not obtained through IMSLP. We included both full scores and study scores, which differ in paper dimensions and therefore can fit different amounts of musical content on a single page. We also included two characteristics that occur in orchestral scores frequently enough to take them into account here. These are the inclusion of choir parts in the score, which adds more text to the score, and the inclusion of non-standard staffs. Generally, staffs consist of five stafflines in CWMN, as can be seen in Figure 1.1, but this is sometimes deviated from for certain instruments. The most common are the percussion instruments, which can be written on staffs of either one or five stafflines. We find this last characteristic especially important to include, since the majority of OMR research does not seem to take this into consideration, though it has been brought up by Fujinaga as early as 1996 [14].

The selected pieces are shown in Table 3.1. This table shows the title and composer of the pieces, as well as the number of pages, number of measures, and number of empty measures for each printed version of the score. The printed scores are all split into single-page PNG images with 300 dpi. Since we want to work with the musical contents only, the auxiliary pages that contain no music, such as covers, table of contents, and additional explanations, were discarded. Furthermore, for two of the scores, we only use a subset of the full score. From Bach's Brandenburg Concerto No. 5 we only use the first movement, since that is the only movement the symbolic score is available for, and furthermore we discard the third and fourth movement of Beethoven's Symphony No. 4, since the symbolic scores for these movements are corrupted. We did make various small changes to some of the Musedata files for the other Beethoven symphonies where we encountered syntax errors and the `music21` toolkit could not parse the files. To validate our previous remark that transcription of these scores could benefit from clustering, we have also indicated the number of empty measures present on each page of the printed score. We see that roughly 33% of the measures are empty measures, so being able to transcribe a large number of them in a more efficient way than processing them on a one-by-one basis would massively reduce the time needed to transcribe a score.

3.2. Score Mapping

Now that all the data has been acquired, the next step is to map the symbolic data to the printed scores. If we have this mapping, we then have a ground truth when comparing measures later on during clustering. When this mapping is done, we need to be able to reliably get the musical information on a measure given a (*page*, *system*, *system measure*, *staff*) tuple, with which we can uniquely identify each measure. This is not a straightforward task, since there is no guarantee that the layout used in the symbolic scores is the same layout used in the printed scores. There are several unknowns when trying to obtain such a mapping. First of all, we need to know which system measures are within any given system. Secondly, we need to find out which parts are shown in this system and in which staffs they are shown. As we saw in Figure 1.1, the parts shown can change per system to save space or gain a better overview. Furthermore, parts are not always mapped one-to-one to staffs in a system. It occurs frequently that multiple parts are shown on the same staff, again to save space. If we can determine these three things, we are guaranteed to have a one-to-one mapping of each measure between the symbolic score and the printed score.

Finding the system measures for a given system is the most straightforward of these tasks. If we know how many system measures each system contains, then mapping the system measures becomes a simple operation of laying the symbolic score over the printed score, starting at the first system measure until the end of the piece, recording which system measures are contained in each system. From that point on, we can determine which parts are shown in each system and in which staffs they appear. We can first leverage the fact that a part always has to be visible in the printed score if that part has any measure in the given system that is not an empty measure. If it does not, musical information would be lost in the printed score. This gives us a subset of parts that are guaranteed to appear in the system. To map these to the staffs, however, we need to know which parts map to the same staff. When we have acquired this information, it will give us the non-empty staffs in the system with the corresponding parts. Comparing the total number of staffs to the found non-empty staffs tells us if there are empty staffs in the system present as well. The only thing we are not certain of is which staffs are empty and which are not. For this, we need information on the order of the parts. With this ordering, we can match the parts to the staffs in most cases by simply going through the parts in order and assigning them to the next staff, taking into account which parts are mapped to the same staff. If there are fewer empty staffs in the system than there are possible empty staffs in the symbolic score, again taking into account part groupings, then it is still not clear which empty staffs are shown in the system. Finding the correct ones is important, since it could affect which parts the subsequent staffs get assigned, so in this case, we need additional information to identify which staffs are empty. With this final piece, we can assign each of the parts to the correct staffs on the page.

3.2.1. Manual Annotations

To obtain all this required information, we made different annotations on the printed scores manually. Each annotation is stored in a text file. These are the annotations we made to solve the problems stated above:

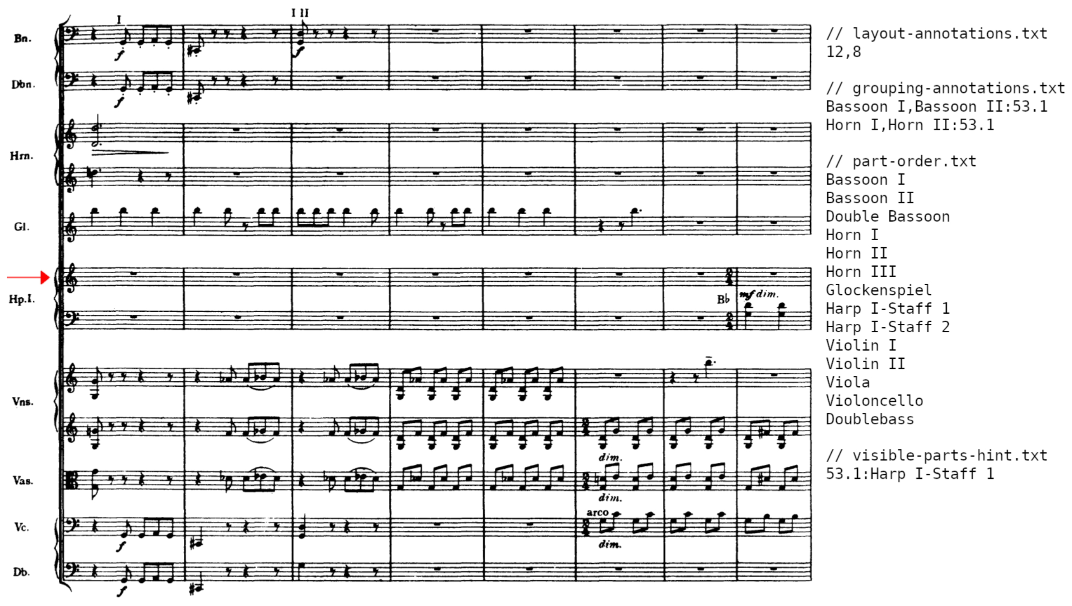
Layout annotations This lists for each system the number of system measures and the number of visible staffs. Each line represents a page and contains a "*<#staffs>,<#system measures>*" tag for each system, separated by whitespaces when a page contains multiple systems. This enables us to map the system measures to the right systems and gives us information on how many staffs are present in a system, which is necessary to determine to which staffs the shown parts need to be mapped.

Grouping annotations This lists which parts are grouped together on a staff. Since the groupings can change per system, part groups are generally specified per page or system. To reduce manual labour, however, it is also possible to provide ranges of pages or to leave out page and system specifications altogether, which groups the parts together for the entire score. An example of a grouping annotation for a score is "*<part1>,<part2>,...,<partn>:2,5-9,11.2*", which indicates that parts *part1* through *partn* are grouped into one staff on page 2, pages 5 through 9 and system 2 of page 11.

Part order annotations This lists the order of the parts in the printed score. Each line contains the identifier of a part in the symbolic score, in the order that part occurs in the printed score. Generally, part orders do not change in printed music, so this ordering holds throughout the entire score. Grouped parts are often adjacent in the part ordering, but if a grouping annotation would conflict with the part order annotations, the grouping annotation trumps the part order annotations.

Visible parts hint This is used in case it is not possible to deterministically match the parts to the correct staffs in the printed score. This hint can be used to indicate which parts are visible in a given system. This is only necessary for empty staffs, since we already know that non-empty staffs are always visible. Each line is of the form "*11.2:<part1>,<part2>*", indicating that on system 2 of page 11, the parts *part1* and *part2* are visible in the printed score. Using this hint in combination with the grouping annotations and the part order annotations, we can find which parts correspond to which staffs in the system.

The score mapping algorithm takes as input the symbolic score and the aforementioned annotations and determines for each system for each page which parts are visible on which staff. This information is stored



```
// layout-annotations.txt
12,8

// grouping-annotations.txt
Bassoon I,Bassoon II:53.1
Horn I,Horn II:53.1

// part-order.txt
Bassoon I
Bassoon II
Double Bassoon
Horn I
Horn II
Horn III
Glockenspiel
Harp I-Staff 1
Harp I-Staff 2
Violin I
Violin II
Viola
Violoncello
Doublebass

// visible-parts-hint.txt
53.1:Harp I-Staff 1
```

Figure 3.1: An example of annotations required for score mapping. This outtake is page 53, system 1 of Holst's The Planets.

on disk, so that it can later be used when evaluating the clusters created in Chapter 6. We give an example of annotations for a given printed page of music in Section 3.2.2.

Since there is quite some manual labour involved here, we performed a few checks to make sure that we catch as many mistakes as possible. First of all, we do a simple check to make sure that the total number of system measures match between the symbolic score and the layout annotations. If they don't, a mistake has been made and we create the layout annotations again. Secondly, after the algorithm creates the mapping, we list the number of empty staffs per system. We manually check this list against the printed score for each system to see if they correspond. Given the size of this dataset, it is not feasible to check this manually on the measure level, but on the system level, we can relatively easily spot empty staffs. When a mismatch is found, we check the staff count in the layout annotations and the grouping annotations to make sure there is not a mistake in either one of them. Finally, we randomly pick a few pages from each score to compare in full with the generated mapping. With these checks in place, we have greatly reduced the risk of errors in the score mappings. We did come across a case where the information in the symbolic score did not match that in the printed score. We resolved this by changing the symbolic score to match the printed score. Although we are aware this might not always be the correct approach from a musicological standpoint, it does not affect this research, since we are interested in the results of measure clustering, for which it only matters that we have a ground truth, not what the ground truth is.

3.2.2. A Score Mapping Example

To show how the annotations work, we provide an example in Figure 3.1. Here we see system 1 of page 53 of Holst's The Planets from our dataset. The necessary annotations to map the symbolic data to this system are shown on the right side of the image. Note that we left out all annotations that are not directly linked to this system for brevity. For example, the layout annotations for all pages before this would be necessary to identify which system measures of the symbolic data correspond to this system. We see that the layout annotation is *12,8*, indicating that there are 12 staffs and 8 system measures shown in this system. The grouping annotations show which parts are grouped into one staff. In this example, this happens in staff 1, where *Bassoon I* and *Bassoon II* are grouped together, and in staff 3, where *Horn I* and *Horn II* are grouped together. A little expert knowledge is necessary to deduce this, in this case, it can be seen by the fact that two notes are played at the same time and, in the case of the first staff, the *I II* notation above the staff. The part order annotations show the identifier of each part in the symbolic score, ordered by their occurrence in the printed score. Note that we only included the parts that are visible in this system, again for brevity. Finally, the visible parts hint shows that *Harp I-Staff 1*, indicated with the red arrow in the image, is visible on the printed score. Visibility of the other parts on the printed score can be deduced from the fact that they have non-empty measures in this system.

4
Staff Detection

Now that we have the data collected and mapped, the next step is to detect the measures that are contained in the printed scores. Just as was done by Vigliensoni et al. [50], this process is divided into two main steps: first, we find the staffs and then we find the barlines. When we have obtained both the staffs and barlines, the measures on a page can be identified. This chapter will be dedicated to the first step, in the next chapter we will discuss the second step, and Chapter 6 will use these to extract the measures from the printed scores. We will describe the downsides to the Dalitz staff detection algorithm and evaluate its performance on our dataset in Section 4.1. After that, we provide our adaptation to the staff detection algorithm and evaluate that on our dataset as well in Section 4.2.

4.1. The Dalitz Staff Detection Algorithm

The staff detection algorithm by Dalitz et al., as discussed in Section 2.1.1, seems like a logical choice for detecting staffs, given that it is used by Vigliensoni et al. to do the same thing in their measure detector [50]. The algorithm makes two crucial assumptions, however, which makes it not suitable for our purposes. Firstly, it will always enforce that each staff on the page has the same number of stafflines. This number is either set by passing the `numlines` parameter, or it is estimated by the algorithm as the most frequently occurring number of stafflines in the detected staffs as `estimated_numlines`. The only way this algorithm can yield staffs that deviate from this number of stafflines, is if it first detects a staff with at least `estimated_numlines` stafflines, and then in step 7 of the algorithm, removes stafflines that are much shorter than the longest staffline found if `numlines` was not set. Secondly, it assumes staffs cannot consist of only a single staffline. This manifests in step 4, where the algorithm removes staffsegments that have no vertical links, making it impossible to have staffs with a single staffline. Even if they were detected here, they would be removed in step 6 regardless, since the single staffline will be less than either `numlines` or `estimated_numlines`.

Despite these downsides to the Dalitz staff detection algorithm, we will first evaluate its performance on the collected data. Since we have layout annotations on each page indicating how many staffs are present in each system, as described in Section 3.2, we can easily compare the number of detected staffs to the expected number. Although this does not evaluate whether all pixels of a staff are detected properly, we assume that if a staff is detected, most of its stafflines will be detected correctly, given the relatively high-quality scans we have in our dataset. Given that the main application in which this will be used later in this work does not require such pixel-perfect precision, this should yield acceptable results. Furthermore, we could leverage staffline groupings in systems to determine whether a staffline deviates from its expected length, given that its length should equal the other elements in the grouping, and adapt those in a later stage. We run the Dalitz staff detection algorithm through the implementation available in the MusicStaves Toolkit[1] written by the authors for the Gamera Toolkit [13]. We do not leave out the optional `numlines` argument, but pass `numlines=5`, because leaving it out would not yield better results. The algorithm would either set `estimated_numlines=5`, since the majority of CWMN is written staffs with 5 stafflines, yielding approximately the same results, or set `estimated_numlines` to some other value and yield worse results because of it. We confirmed this behaviour during initial testing. An example of detected staffs overlayed over their original page can be seen in Figure 4.1. The staffs are separated visually from each other by alternating colours.

[1] `http://music-staves.sf.net/`

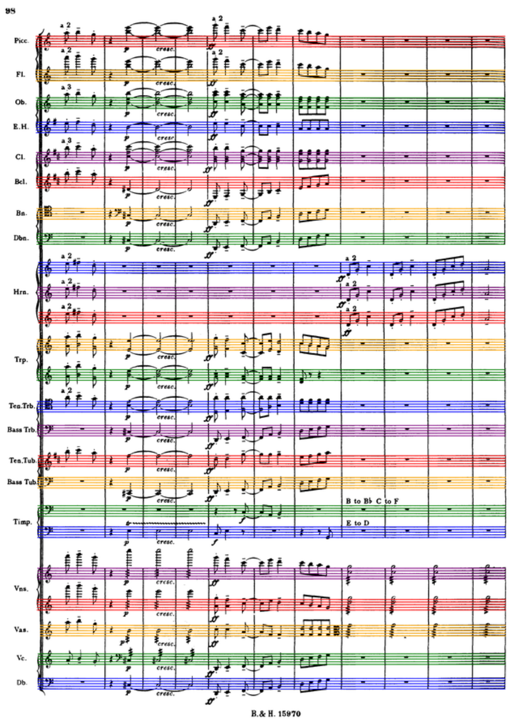


Figure 4.1: An example of staff detection results, overlayed over their original page.

The number of detected staffs can be easily compared to the annotations that were made for each page. For each page, the performance of the algorithm is determined as 1 minus the absolute error, with which the performance over an entire score can be calculated by 1 minus the mean absolute error. The performance metric for a score is then:

$$Perf = 1 - MAE = 1 - \frac{1}{n}\sum_{i=1}^{n}\left|\frac{A_i - F_i}{A_i}\right| \tag{4.1}$$

where n is the number of pages in a score, A_i is the true number of staffs on page i and F_i is the found number of staffs on page i. The scores are evaluated per page, since that is the most granularity we can provide with our annotations. Calculating the absolute error over several pages simultaneously could possibly lead to errors cancelling each other out. Ideally, we would like to provide even more granularity to avoid errors cancelling each other out within the same page, but creating manual annotations on such a small scale is unfeasible on a data collection this large. The resulting performance for each score can be found in Table 4.1. The results of our adaptation are also included in the table, for an overview of the adaptation, see Section 4.2.

The performance scores are already quite high, never dipping below 0.93. Therefore, we categorize the errors to gain a better understanding of how these errors are made. To facilitate this, we overlay the staffs over each original page, and visually inspect the pages on which the detected number of staffs deviates from the corresponding layout annotations. We divide the errors made by the staff detection algorithms into the following six categories.

A. Staffs consisting of a single staffline are not detected.

B. Staffs are not detected or multiple staffs are detected as one.

C. One staff is detected as multiple staffs.

D. Horizontal elements, such as slurs, ties, trills, voltas, dynamics, etc. are detected as staffs.

E. An ossia, or auxiliary staff, is detected.

F. The quality of the scanned page is too poor for reliable staff detection.

Table 4.1: *Perf* per score for both the Dalitz and the adapted staff detection algorithms.

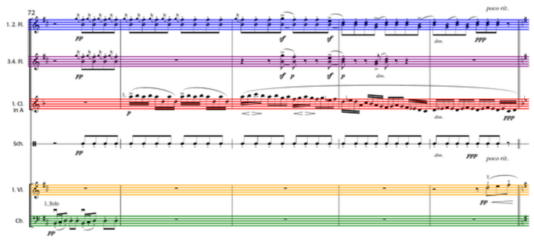
Score	**Composer**	**Dalitz** *Perf*	**Adaptation** *Perf*
Brandenburg Concerto No. 5	Bach, J.S.	0.9804	1.0
Symphony No. 1	Beethoven, L. van	0.9804	1.0
Symphony No. 2	Beethoven, L. van	0.9894	0.9993
Symphony No. 3	Beethoven, L. van	0.9821	0.9943
Symphony No. 4	Beethoven, L. van	0.9862	0.9882
Symphony No. 5	Beethoven, L. van	0.9968	0.999
Symphony No. 6	Beethoven, L. van	0.9908	0.9994
Symphony No. 7	Beethoven, L. van	0.987	0.9995
Symphony No. 8	Beethoven, L. van	0.9952	0.9986
Symphony No. 9	Beethoven, L. van	0.9791	0.9954
Symphony No. 3	Brahms, J.	1.0	1.0
Symphony No. 5	Bruckner, A.	0.9746	0.9966
Symphony No. 9	Bruckner, A.	0.9995	0.997
The Planets	Holst, G.	0.9624	0.9941
Symphony No. 4	Mahler, G.	0.9625	0.9967
Symphony No. 41	Mozart, W.A.	0.9982	1.0
Ouverture 1812	Tchaikovsky, P.I.	0.9303	0.9992

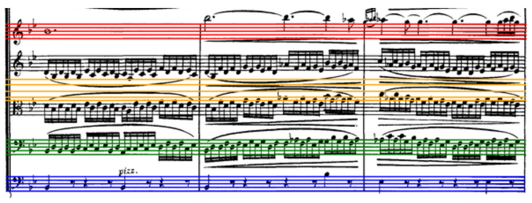
Examples of all of the categories can be found in Figure 4.2. Category A is a logical first category, since this is one of the error types the Dalitz algorithm is expected to make. Category B deals with too few staffs that are detected on a page or a section of a page. It can either be that a staff is not detected at all, but its neighbours are, or that multiple staffs are detected as one. Category C specifies errors that are made the other way around, where a single staff is detected as multiple staffs. Category D describes non-staff elements that are detected as staffs incorrectly. These are mainly long horizontal elements that can occur in music scores. A category E error is made when an ossia is detected. An ossia is used in sheet music to depict alternative suggestions to the written music. This is often done in the form of a smaller staff, above or below the place of interest, or on the top or bottom of the page. Although an ossia does depict a staff and therefore the algorithm detects it correctly, it is not reflected in the symbolic music data and so we classify this as an error in this work. Finally, category F collects the pages where scans are of such poor quality that staff detection algorithms based on image processing techniques can not reasonably be expected to detect the staffs.

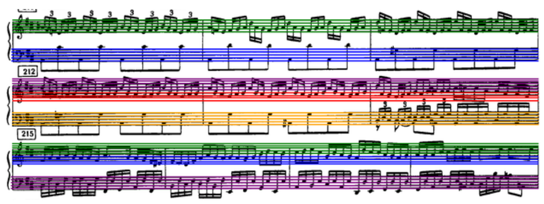
Table 4.2: Total number of errors per category for both the Dalitz and the adapted staff detection algorithms.

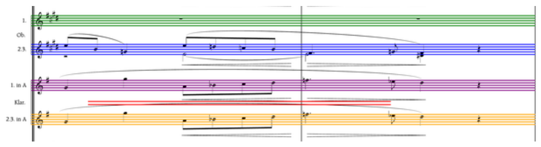
Category	**Dalitz errors**	**Adaptation errors**
A	73	0
B	209	19
C	32	2
D	15	40
E	1	1
F	4	2

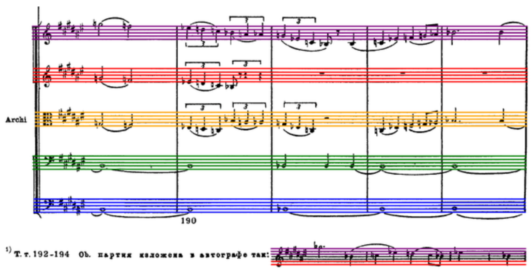
We have counted the number of occurrences of errors in each category, the results of which can be seen in Table 4.2, which again also includes the results of our adaptation of Section 4.2. Errors of category A are of course to be expected, given the assumptions the Dalitz algorithm makes. However, not all single staffline staffs are ignored. Some of them are detected, in which case the algorithm will force too many stafflines to be detected for this staff to adhere to `numlines=5`, of which an example can be seen in Figure 4.3. What is more surprising is the number of occurences of errors of category B. In these real-world scores for larger ensembles, the staffs are generally spaced much closer together than in the dataset used by Dalitz et al. Together with `numlines=5` that is being enforced, this can cause the algorithm to group the stafflines of two subsequent staffs together into one staff, and remove stafflines until only 5 are left. This either causes a staff to be missed because all its stafflines are discarded, or two staffs to be detected as one because a subset of each of their stafflines is used to construct one staff. There are a few instances of error category C, where one staff is detected as multiple staffs. This can happen when one of the stafflines of a staff is not detected properly,


(a) A: Staffs with only one staffline are not detected.


(b) B: Multiple staffs are detected as one, or staffs are not detected.


(c) C: One staff is detected as multiple staffs.


(d) D: Horizontal non-staff elements are detected as staffs.


(e) E: An ossia is detected.

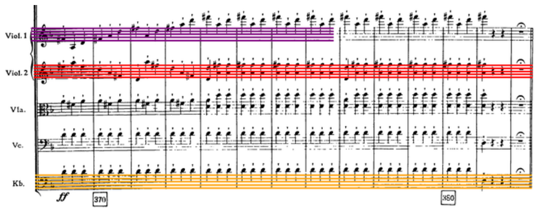

(f) F: Poor quality of input page prevents proper staff detection.

Figure 4.2: Examples of each of the staff detection error categories.

or when the skeletonization algorithm produces *staffsegments* from stafflines that are too far apart vertically to be linked in step 3, both of which cause a group of *staffsegments* belonging to the same staff to not be linked vertically. This results in two separate staffs, each of which will be forced to have the predetermined 5 stafflines. Category D has a few occurrences, spread out over the available scores. There is one ossia in the dataset that is correctly detected as being a staff, but since this does not concur with the symbolic data, we count is an error in category E here. Finally, in category F there are 4 pages of music where the quality of the scans is very poor.

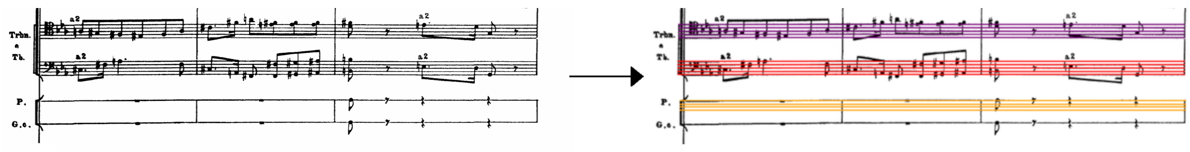


Figure 4.3: The Dalitz algorithm will sometimes detect staffs of only one staffline, but force them to have too many stafflines.

4.2. An Adaptation to Staff Detection

To overcome the errors made in the staff detection algorithm by Dalitz, we propose our own adapted version of the algorithm. Its main goal is to eliminate the assumption that staffs can not consist of a single staffline, but we also attempt to make the adapted version more robust against the other errors the Dalitz algorithm makes. The adapted algorithm is described below.

1. Take as input the original image.
2. Obtain the *staffsegments* by extracting the horizontal runs, thinning the runs vertically, as described

in Algorithm 1, with `window_size` set to 3 × *staffspace-height*. Remove the *staffsegments* that are too short.

3. Add vertical links between *staffsegments* that overlap horizontally and have a vertical distance around *staffspace-height* + *staffline-height*.

4. Each group of connected *staffsegments* is now considered a staff. The staffs are labelled, and each *staffsegment* in that staff is labelled based on the number of vertical link steps that precede it. Now each staffline consists of several *staffsegments*.

5. Within each staff, combine the *staffsegments* that overlap to make sure no *staffsegments* in each staffline overlap.

6. Remove all stafflines that are much shorter than the longest staffline. The threshold is set at 0.8 times the length of the longest staffline.

7. Remove staffs that horizontally overlap with wider staffs. If any of the remaining staffs are sufficiently close together horizontally, merge them.

8. If any stafflines are broken, interpolate between the *staffsegments* to join them together.

9. Rectify any incorrect staff splits or joins. Go through the detected staffs from top to bottom and modify any existing staffs based on the vertical distance between the individual stafflines. If two subsequent stafflines within a staff are more than 3 × *staffspace-height* apart, split the staff into two. If two subsequent stafflines between two staffs are less than 1.5 × *staffspace-height* apart, join the two staffs.

There are a few differences when comparing it to the Dalitz algorithm. First of all, the optional `numlines` argument has been removed, since we do not assume that all staffs have the same number of stafflines. Secondly, step 4 does not remove stafflines that have no vertical links anymore. This step is the main reason that the Dalitz algorithm does not detect staffs with only one staffline, since those would not have vertical links to other stafflines. Step 6 of the Dalitz algorithm has been removed in its entirety, since that removes any staffs that do not adhere to `numlines` and that option has been removed. The previous step 7, which is now step 6, has been simplified, since the `numlines` argument has been removed: it now only removes any stafflines that are significantly shorter than the longest one in that staff. Finally, a new final step has been added, which serves as a last sanity check. It will attempt to rectify any mistakes that were made during the grouping of the stafflines by joining staffs that are very close together vertically or splitting up staffs if two subsequent stafflines are far apart vertically.

With this new staff detection algorithm, we execute the same steps as we did with the Dalitz algorithm. We run the algorithm on each page, evaluate its results by comparing the found staffs to the layout annotations to determine *Perf*, and categorize the errors in the previously established categories through visual observation. The results are included in Tables 4.1 and 4.2 so that they can be compared to the results of the Dalitz algorithm. In Table 4.1, we see that our adaptation provides a clear improvement over the Dalitz algorithm. Only in Bruckner Symphony No. 9 does the Dalitz algorithm slightly outperform our staff detection algorithm. When we look at the different types of errors that are being made, we see that all errors of category A have been eliminated, which means that the main goal of our adaptation has been reached. Furthermore, we see a very large decrease in errors of category B. This confirms our hypothesis that the fixed `numlines` parameter actually prevents the Dalitz algorithm from detecting all staffs reliably. Now that this assumption has been removed, more staffs can be detected properly. One could expect that this would increase the errors of category C, but the number of errors in this category have actually been reduced significantly as well. The final step that has been added to our adaptation has a large hand in reducing the errors of categories B and C. The only downside of the adaptation is that errors of category D have gone up. This is a logical result of removing the assumption that staffs with only one staffline can not occur. With this assumption removed, more long horizontal elements are now misidentified as stafflines, which results in too many detected staffs. However, this does not weigh up against the improvements that have been obtained with these adaptations for the purposes of this work. Our adaptation detected the same ossia that was detected by the Dalitz staff detection algorithm, as can be seen in category E. Surprisingly, it has detected the staffs on 2 of the 4 pages which we classified as “poor quality”. On one of those pages, however, some of the detected staffs are significantly shorter than they should be, so it is not entirely accurate to say that the staffs have been detected properly.

This page was intentionally left blank.

5
Barline Detection

In the previous chapter, we outlined the steps we took to detect the staffs in the music scores and evaluated those algorithms. In this chapter, we will discuss the second component needed for measure detection: detection of the barlines. We discuss the considerations taken into account when designing an algorithm for this in Section 5.1, the barline detection algorithm itself in Section 5.2, and evaluate the results of the barline detection algorithm on our dataset in Section 5.3.

5.1. Preliminary Considerations

As was covered in Section 2.2, the work by Vigliensoni et al. [50] attempts to detect system measures on a page of music. A system measure could be combined with the staffs detected using our adaptation of the staff detection algorithm to find the measures on a page. However, the Vigliensoni measure detector requires human input for each page that is processed, which costs time and energy that can quickly add up in larger musical works and is prone to human errors. Therefore, we developed a new barline detection algorithm that attempts to identify the barlines on a page of music without the need for human input. For this design, we take the same general approach as we did for staff detection, by first detecting appropriate segments on the page and then grouping them together into the required objects. The detected barlines are then combined with the detected staffs in the next chapter to complete the measure detection algorithm.

Three things need to be taken into account when designing this barline detection algorithm. Firstly, a barline is not always a continuous vertical line throughout an entire system, see Figure 5.1 for an example. There, we see two systems, each of which have barlines that are split into three separate segments. One such barline in the first system is highlighted in the image. These segments are part of the same barline and need to be identified as such to make make sure the system measures are detected properly. The important assumption we can make, however, is that each system will always have at least one barline that spans the entire height of the system. Without such a barline, an unambiguous separation between systems would not be possible. This brings us to the second consideration. After we have detected the staffs and barlines, they need to be divided into systems. This is something that could be done in a later stage, especially since the detected staffs are not yet divided into systems themselves. Given our assumption that each system will have at least one full-height barline, however, it makes sense to incorporate the separation of systems into this step. Finally, it is possible that in some places a double barline is drawn on the page instead of a single one. The left-most barlines of both systems in Figure 5.1 are an example of that. These double barlines are characterized by the fact that they consist of two vertical lines that are very close together, and should be treated as if they were a single line for the measure detection results to make sense.

5.2. The Barline Detection Algorithm

With these considerations in mind, we now develop the barline detection algorithm. This algorithm takes an approach that corresponds with the staff detection algorithm, by starting with the same skeletonization step. However, since we are in this case interested in vertical segments instead of horizontal ones, we perform the skeletonization on a 90° rotated version of the page image. The obtained segments are then processed in a number of steps to find the final barlines, similarly to how this is done in the staff detection algorithm. We describe the steps taken here:

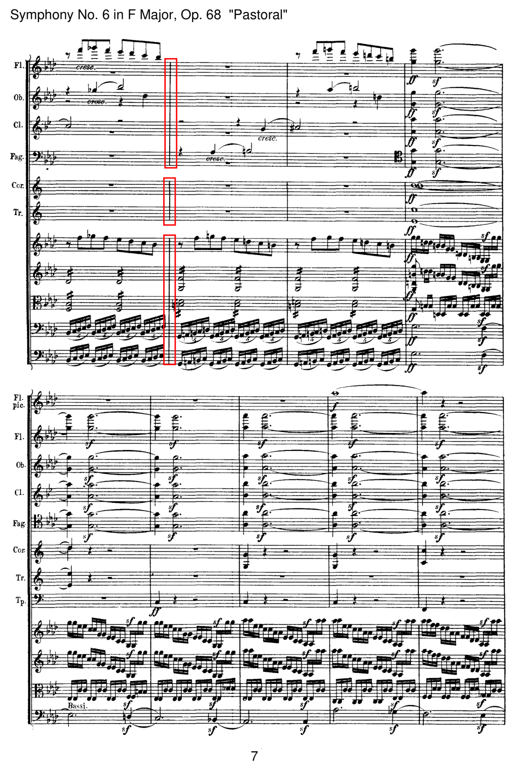


Figure 5.1: A page of music where the barlines are broken up into multiple segments. The segments of a barline in the first system are highlighted.

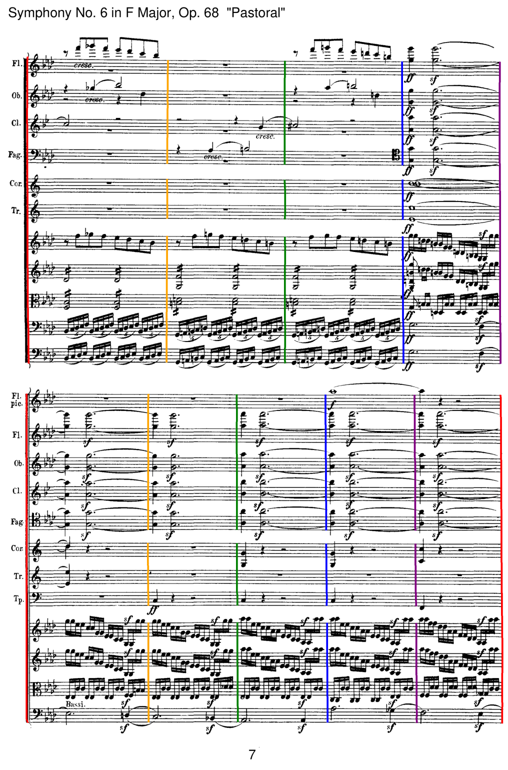


Figure 5.2: A page of music with the detected barlines highlighted.

1. Take as input the original image.

2. Obtain barline segment candidates by extracting the horizontal black runs and thinning the runs vertically on a 90° rotated input, as described in Algorithm 1. Rotate the output image back to obtain vertical segments.

3. Remove all candidate segments that are too short. The threshold is set at 2 × *staffheight*, assuming 5 stafflines (*staffheight* = 5 × *staffline-height* + 4 × *staffspace-height*).

4. Group all segments into systems. This is done by grouping two segments or two groups of segments together if their projections onto the y-axis overlap, until no overlap can be found between any groups of segments. When two groups of segments have no overlap on the y-axis projection, they are considered to be separate systems. This procedure can be easily done by sorting on the vertical position of the segments and starting with a group containing only the first segment. Each next segment is added to the group while the segment overlaps with the group. If the next segment has no overlap with the group, a new group is started with that segment, to which all subsequent segments are assigned until no more overlap is found. This repeats until all segments are assigned to a group.

5. Within each group, create groups of segments that are likely to represent a single barline. This is done by comparing the segment positions and orientations: if two segments do not overlap vertically, and their horizontal positions are close together, and their orientation is approximately the same, they are likely to be part of the same barline, and so they are grouped together. Each of these groups is now considered to be a barline.

6. Remove barlines that are too close to another barline. The distance is only measured in the horizontal direction. If two barlines are too close together, the longest of the two is kept.

7. Construct a full-height barline from the segments of each barline by interpolating the areas in between segments. All black runs that were found in step 2 that fit this interpolated barline within a certain

margin are selected as segments of this barline. These selected segments are now the only parts that make up the barline.

8. Remove barlines that are either much shorter than the second-longest barline and barlines of which the segments do not cover at least half of the height of the barline.

9. Remove systems that do not have at least two barlines detected.

There are a few things to note about these steps. In step 2, the `window_size` parameter of the skeletonization algorithm is set to 5 × *staffspace-height*, which is larger than the window used in the staff detection algorithm. A smaller `window_size` is suitable for staff detection, but we found that for detection of vertical black runs in barline detection, a larger value would generate less noise to sift through in later steps, while still detecting the required segments properly. In step 5, both the position and orientation of the segments are taken into account. The position is checked to make sure the segments do not overlap vertically, meaning they would be placed next to each other, and do overlap horizontally, meaning they are placed below each other, which is the only configuration in which they could not be part of the same barline. Orientation is also checked as a fail-safe. Although the skeletonization algorithm yields only vertical segments, after undoing the 90° rotation, it does try to follow any lines, and so segments are not always perfectly straight. If two segments could be grouped together based on their positions, but their orientations differ, they are most likely not part of a single barline and thus should not be grouped together. Step 6 is taken to avoid multiple detections of a double barline, which we discussed in Section 5.1. Step 7 is necessary to include black runs that are part of the barline, but do not meet the length threshold set in step 3. Step 8 filters out the most common false positives. In this step, we chose to compare the lengths of the barlines to the second longest barline, because the longest barline can sometimes be different in height than the other barlines. This first barline of a system can often have ornamentations, which makes it stand out a bit more to accommodate easy identification of systems by the reader. This does however mean that these barlines can be longer than all other ones in the system. Instead of relaxing the margin within which the lengths of the other barlines should be, we opted to compare the barlines to the second longest barline. In case the first barline has ornamentations, we do not create false negatives, and in case it does not, the lengths of the two longest barlines are comparable, so it makes little difference to which barline we are comparing. Finally, step 9 is a sanity check that removes detected systems that cannot exist, since a system measure needs at least two barlines to exist.

An example of detected barlines is shown in Figure 5.2. Note that the barlines are divided into two systems, with each system using the same colour combinations to highlight their barlines. Furthermore, the left-most barlines are only detected once, and for all other barlines, the individual barline segments are detected properly.

Table 5.1: *Perf* per score for the barline detection algorithm.

Score	**Composer**	**Barline** *Perf*
Brandenburg Concerto No. 5	Bach, J.S.	0.9726
Symphony No. 1	Beethoven, L. van	0.9799
Symphony No. 2	Beethoven, L. van	0.9888
Symphony No. 3	Beethoven, L. van	0.9753
Symphony No. 4	Beethoven, L. van	0.9813
Symphony No. 5	Beethoven, L. van	0.9865
Symphony No. 6	Beethoven, L. van	0.9893
Symphony No. 7	Beethoven, L. van	0.9855
Symphony No. 8	Beethoven, L. van	0.9955
Symphony No. 9	Beethoven, L. van	0.9746
Symphony No. 3	Brahms, J.	1.0
Symphony No. 5	Bruckner, A.	0.9994
Symphony No. 9	Bruckner, A.	1.0
The Planets	Holst, G.	0.9865
Symphony No. 4	Mahler, G.	0.9996
Symphony No. 41	Mozart, W.A.	0.999
Ouverture 1812	Tchaikovsky, P.I.	0.9839

5.3. Evaluation

Now that we have the barline detection algorithm, we run it on our dataset to obtain all the barlines, and we evaluate its results. We do this in the same manner as we did with both staff detection algorithms, using the layout annotations of Section 3.2. We first compare the number of detected systems to the annotations, since we can only compare the detected number of barlines per system if the number of systems is equal. If the number of systems would not be equal, it would be ambiguous which system needs to be compared to which layout annotation. Once we know that the number of detected systems is correct, we compare the detected barlines to the layout annotations.

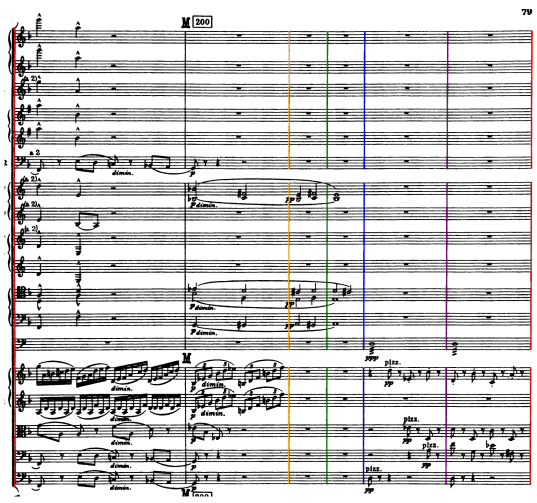
We calculate *Perf* from Equation 4.1 in the same manner as we did for the detected staffs, but here A_i and F_i refer to the actual and found number of barlines per system, instead of staffs per page. The actual number of barlines A_i is given by $A_i = L_i + 1$, where L_i indicates the number of system measures as per the layout annotations. We have first verified that all the systems are detected correctly, so that we can calculate *Perf* without having to worry about layout annotations that cannot be applied to a page unambiguously. The results can be seen in Table 5.1. Analogous to the staff detection evaluation, we divide the observed errors into categories to get a better understanding of what the barline detection algorithm has problems with. We separate the errors into the following 4 categories:

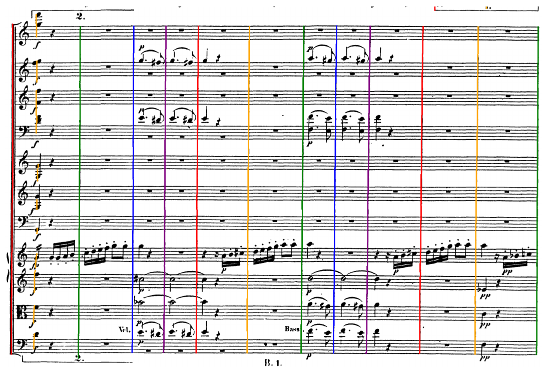
A. Barlines are not detected properly.

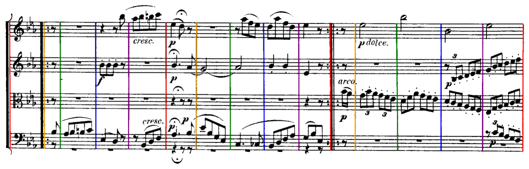
B. Vertical elements, such as note stems, rests, rectangular vertices, etc. are detected as a barline, often when they are stacked on top of one another.

C. A barline of a non-measure was detected properly.

D. A barline was detected due to noise on the page.


(a) A: A barline is not detected properly.


(b) B: Vertical elements, such as notes stacked on top of each other, are detected as barline.


(c) C: A barline of a non-measure is detected when e.g. a repeating section starts.

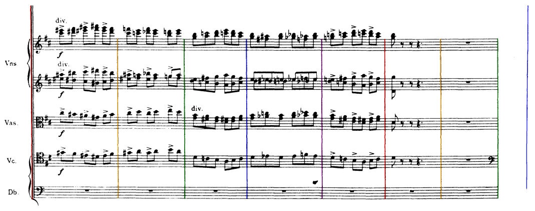

(d) D: False positives due to poor scan quality.

Figure 5.3: Examples of each of the barline detection error categories.

Category A describes barlines that are not detected due to any number of reasons. It is possible that it cannot be differentiated from the surrounding musical objects, or that it was removed in step 8 of the barline detection algorithm. Additionally, poor print quality or excessive curvature can be a reason. Category

B collects all false positives that occur because of other symbols in the music. A large number of vertical elements which are stacked vertically within a system could cause a barline to be incorrectly detected. Step 8 of the algorithm removes a lot of these occurrences, because often these detected barlines are either much shorter than the actual barlines, or because the segments in these barlines are very sparse. It cannot remove all of them reliably without causing false negatives, however. Category C describes the detection of lines on the page that are drawn in the same way as barlines, but do not separate system measures. These can be used to e.g. indicate repeating sections in music. If the repeating section begins at the start of the system, sometimes first a barline at the very start is drawn, then the clefs, accidentals, and time signatures are placed on the staffs, and then an additional "barline" is drawn to signify the start of the repeating section. Both the barline at the start and the "barline" are detected, which is a logical consequence of our algorithm, but does not conform to the symbolic data, since that only records a single barline. Finally, category D contains all false positives that are detected from noise due to poor scan quality. This is in contrast to the false positives of category B, which are not due to noise, but due to other symbols. Examples of each of the error categories can be seen in Figure 5.3. Note the notes stacked together in the first system measure of Figure 5.3b which causes a false positive, and the noise generated by the scanner to the right of the system in Figure 5.3d.

Table 5.2: Total number of errors per category for the barline detection algorithm.

Category	**Errors**
A	122
B	41
C	4
D	8

The results are shown in Table 5.2. We see that especially errors of category A occur most frequently, meaning we have false negatives. On the other hand, we also have a substantial number of false positives in category B. During experimentation, we found that these two categories form a balance when the margins used in step 8 are tweaked. Such a balance is not desirable, since that would mean we could not improve on this further with the current algorithm. We elaborate more on this in Chapter 7. Furthermore, there are only a few errors of categories C and D. Category C errors are false positives that we cannot easily fix with this algorithm, given that these "barlines" are visually a match to the barlines we want to detect, they just do not match in function. A post-processing step would be needed to remove these barlines from the results. The errors in category D all occur in Holst's The Planets and are very similar to the example given in Figure 5.3d, where an additional barline outside of the system boundaries is detected due to poor scan quality. We also propose a solution for this in Chapter 7.

This page was intentionally left blank.

6
Measure Clustering

Now that we have worked out staff detection in Chapter 4 and barline detection in Chapter 5, we can move on to measure clustering. We show how to extract the measures based on the results of the staff and barline detection steps in Section 6.1. Next, we will cover distance calculations between measures in Section 6.2. In Section 6.3, we discuss how to cluster those measures, and finally, in Section 6.4 we will evaluate the quality of these clusters.

6.1. Combining Staffs and Barlines

Given the detected staffs and barlines, we can now extract the measures from the input images. However, the detection of staffs and barlines gives imperfect results, as we have shown in the previous chapters. To avoid these imperfect results influencing the next steps, we remove the pages for which the found number of staffs and barlines does not match with the layout annotations of Section 3.2. This removes 196 out of the total 1764 pages, meaning that 88.9% of the pages remain for our measure clustering experiments.

The grid that is formed by combining the staffs and barlines define the measure boundaries. Each measure is bounded by two barlines in the horizontal direction and by the limits of one staff in the vertical direction. However, this leaves space in between staffs which is not included in measures. Musical contents frequently overflows into this area, so the boundaries of the measures need to be extended vertically. This extension is not trivial, unfortunately. The area in between two staffs is disputed territory, which both the staff above and below can use as a spill-over area simultaneously. The division between the symbols belonging to the staff above and the symbols belonging to the staff below can even be non-linear, as can be seen in Figure 6.1. In this example, the dynamic *fortissimo*, indicated by the ***ff*** symbol, and the note directly to the right of it have overlapping y-coordinates, but the dynamic belongs to the upper staff, while the note belongs to the lower staff, meaning no linear division is possible between the two measures. Even when a linear division is possible, it is not always clear which symbols belong to which staff, since they do not necessarily have to be connected to the staff. We would require reliable symbol recognition to make this division properly. For these reasons, most literature will use the staff itself as the bounding box of the measure, as can be seen in e.g. the work by Zalkow et al. [52]. However, to include as much of the available musical information as possible for the clustering algorithm, we choose to simply divide the space in between measures equally between the two measures. Although this is an imperfect solution, as we explained above, this means that there is more information available for the clustering algorithm than when only the staff boundaries are used. See Figure 6.2 for an example of the measure boundaries we use. We extract an image for each individual measure using these boundaries. These measure images are the objects that will be used in the next sections.

6.2. Distances between Measures

Now that we have images of each individual measure, we need a distance metric to quantify the distance between two measures. Out of rhythm and pitch, the two main components of music, we have opted to focus on the former one. In written music, shifts in pitch only move the note vertically, but changes in rhythm actually dictate how many notes are written. This makes it so that rhythmic differences have a larger impact on the visual differences between two measures. Also, when there is a rhythmic difference between measures, comparing pitch between the measures is ambiguous, since it is not clear which notes need to be compared

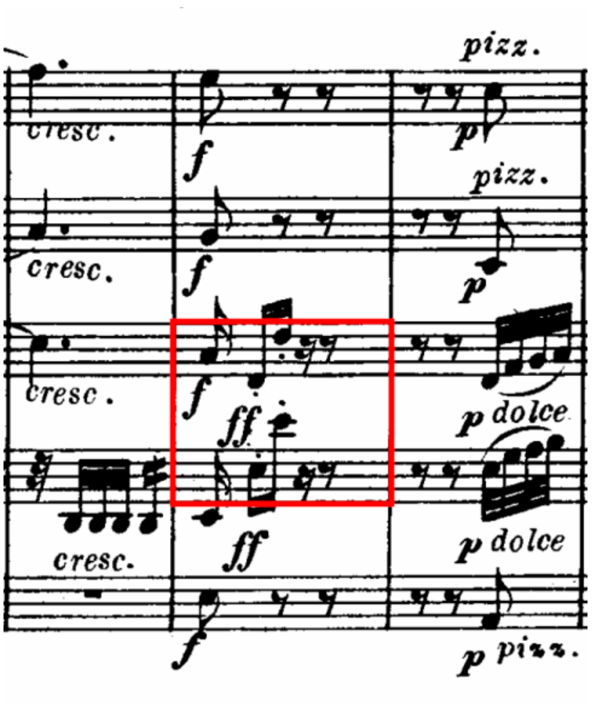


Figure 6.1: An example of non-linear division between two measures. The two measures highlighted in red share the space in between them.

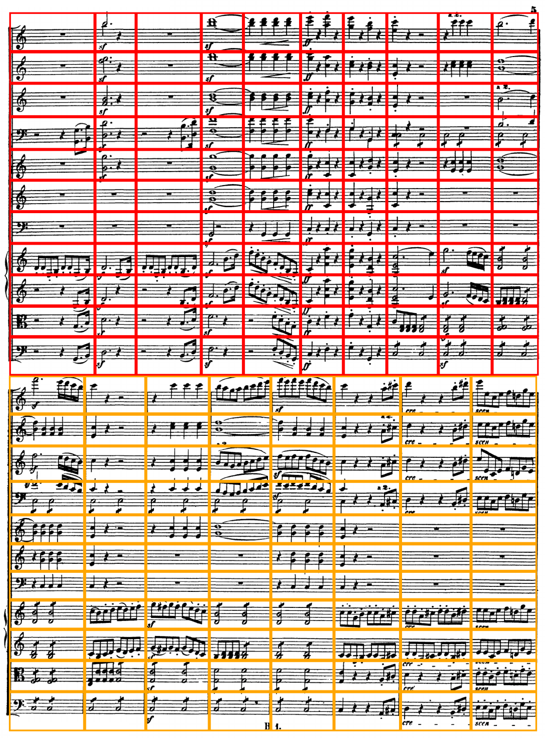


Figure 6.2: Boundaries of each measure on an example page. These boundaries are where the images for each individual measure will be cut off.

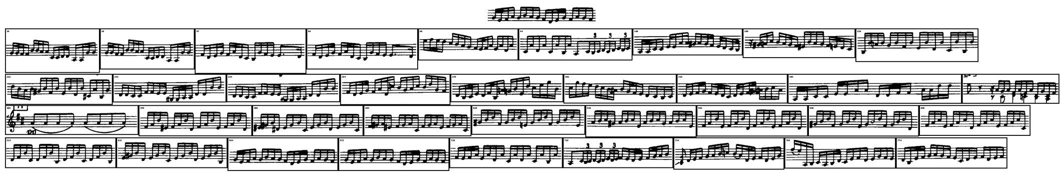
to one another. Furthermore, it frequently happens that rhythmically equal measures occur multiple times in the same piece of music, but they can be transposed, meaning they are shifted in pitch. This can happen either because the rhythm of the measure is very simple, and is therefore an often-used building block in the music, or because a composer has written a reoccurring theme, which can return at a later time in the music, either in the same pitch or transposed. These reasons indicate that it is beneficial to choose rhythm as the component of interest for our similarity study.

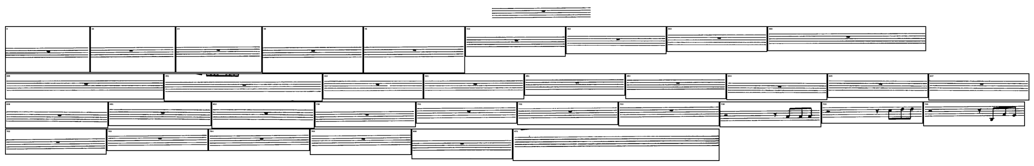
To find rhythmic similarities, we use the Dynamic Time Warping algorithm, which we discussed in Section 2.3. We can represent an image of a measure as a time series by using the intensity profile of the image when projecting it onto the x-axis. Since changes in rhythm would mean adding or removing notes, those changes would manifest mostly in the horizontal direction. This means that a projection onto the x-axis can capture these differences. These profiles are of differing lengths, since the measures are of differing lengths, and so DTW is a suitable metric for this purpose. Note that DTW is not actually a metric, but a measure, since it does not adhere to the triangle inequality, but as we explained in Section 1.8, we use the term metric incorrectly here, since the term "measure" refers to the musical unit instead of a measurement in this work.

Before we obtain the intensity profiles of the measures, we first remove the barlines from the original image. Measures are not always accurately extracted at the barlines, since we use rectangular bounding boxes. This could cause (parts of) barlines to be present in the measure images, causing high peaks in the intensity profiles where no rhythmic information is conveyed. To avoid including these high peaks in the intensity profile, we remove the barlines prior to extracting the measure images. With the intensity profiles obtained, we can determine the distance between each pair of measures in a score. We use the FastDTW algorithm [40] to find the distances and construct a distance matrix with these results. Since we have a very large number of measures in our dataset, this task takes a long time, even though the FastDTW algorithm has been optimized over the original DTW algorithm. The runtime of the distance calculation of all measure pairs can be made manageable through parallelization, as the distance between each pair of measures can be determined independently from all other calculations. Even so, the distance calculations were the main computational bottleneck of this thesis.

6.3. Building Clusters of Measures

With the measure images extracted and the distances calculated, we can move to clustering the measures based on their distances. We use the FastPAM clustering algorithm [42], which we also discussed in Section 2.4. This algorithm takes the distance matrix from the previous section and the desired number of clusters k as input, and yields k clusters which each have a medoid and a list of measures assigned to it. We obtain the clusters for $k \in \{2 \ldots 150\}$. Since the algorithm has a random initialization and finds local optima, we rerun the algorithm 10 times for each setting. The result with the smallest recorded TD is used as the clustering for that k. For each k, we record the loss and construct an elbow plot. The Kneedle algorithm [41] is used to select the optimal k. The selected number of clusters can be found in Table 6.1, together with the number of classes in each score, which we discuss in the next section. The clustering result that was used to calculate the minimal loss for the selected k is used as the definitive clustering for that score. Three such clusters from Bach's Brandenburg Concerto No. 5 can be seen in Figure 6.3. The medoid of each cluster is shown at the top of the image and all the measures of the cluster are stitched together after that. In these few cases, the clustering algorithm seems to be performing reasonably well when inspecting the results visually. The measures of e.g. a cluster with a more involved rhythm as in Figure 6.3a are separated from a cluster of mainly empty measures in Figure 6.3b.


(a) Cluster 3 of Bach's Brandenburg Concerto No. 5


(b) Cluster 6 of Bach's Brandenburg Concerto No. 5

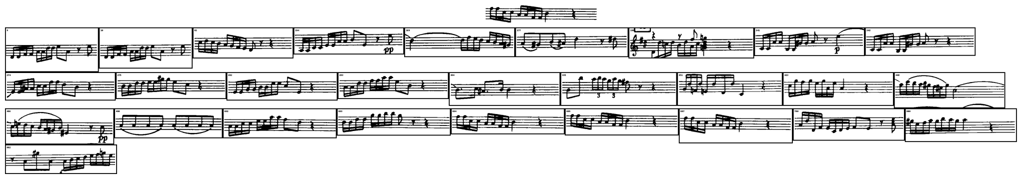

(c) Cluster 7 of Bach's Brandenburg Concerto No. 5

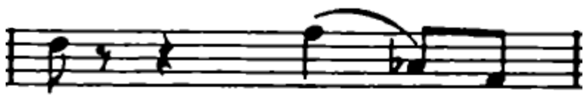
Figure 6.3: Three clusters from Bach's Brandenburg Concerto No. 5. The medoid of each cluster is shown at the top, after which all the measures in that cluster are shown.

6.4. Evaluation of the Clusters

Finally, we need to evaluate the quality of the clusters we have found. To do this, we make use of the mapping of our symbolic data to our printed scores from Section 3.2. The mapping gives us access to the actual musical information in a measure, and so we can compare the measures within a cluster to each other to determine whether we actually succeeded in grouping measures with the same rhythm in the same cluster. To this end, we encode the rhythmic information within a measure in a similar way as was done by Mongeau and Sankoff [25], which has a resemblance to the MIDI specification [45]. Each quarter note is given a fixed number of pulses per quarter, *ppq*, which we set to 24. This is the minimum *ppq* that can be set in MIDI, and we use it as well since it is divisible by a large number of frequently occurring note lengths. Each note or rest is then encoded as a $(start, duration, isNote)$ tuple. The *duration* of each note or rest is determined as

$$duration = \frac{denominator}{4} \times quarterLength \times ppq \tag{6.1}$$

where *denominator* is the denominator of the currently applicable time signature, and *quarterLength* is the length of the note or rest, expressed in quarter note lengths. The $\frac{denominator}{4}$ term serves as normalization

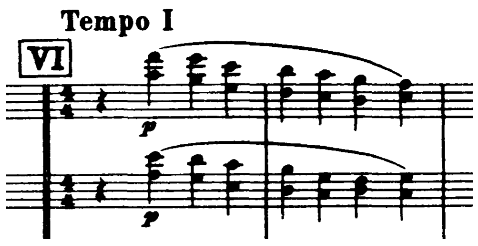


```
[
  (0 , 12, 1), (12, 12, 0), (24, 24, 0),
  (48, 24, 1), (72, 12, 1), (84, 12, 1),
]
```

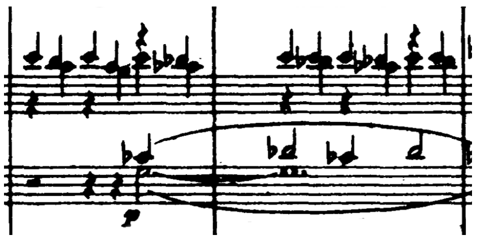
Figure 6.4: An example of the encoded rhythm of a measure.

between different time signatures, to make sure that rhythms written in different time signatures are encoded in the same way if they convey the same rhythm. The *start* element of the encoding specifies the placement of the note or rest in the measure. The first note is placed at *start* = 0, and the *start* for each subsequent note or rest is determined by adding the duration of the previous note to that previous note's *start* element. The *isNote* element of the encoding is a boolean value that differentiates between notes and rests. Gathering the encodings of all the notes and rests in a measure in a list gives the encoding of a measure. As an example, we encoded one such measure in Figure 6.4.

This works well for monophonic measures, where only one note at a time is played, but special care needs to be taken for polyphonic measures. Polyphony occurs when multiple voices play in one measure. This can occur either when multiple parts are shown in the same staff, or when a polyphonic instrument plays chords, which for the purposes of this work we also consider to be multiple voices. We distinguish two types of polyphonic measures: type 1 polyphonic measures have polyphonic voices that all have the same rhythm, whereas type 2 polyphonic measures have polyphonic voices that differ in rhythm from each other. We give an example of each type in Figure 6.5. The first type of polyphony results in little extra information shown in the measure, except for additional note heads. The note stems and rests will all be shared by the voices. Therefore, such a measure is encoded as if it is a monophonic measure, as there should not be much difference between the images of such a polyphonic measure and a monophonic measure with the same rhythm. Type 2 polyphonic measures have extra notes or rests present in the measure, however, and so we encode each of them separately. This means that the rhythmic encoding of type 2 polyphonic measures can consist of much more entries relative to monophonic measures with similar rhythms. We think this is a suitable solution for our research purposes, as we expect that most of the benefits from finding similarities can be obtained from simple rhythmic similarities, which occur mostly in monophonic or type 1 polyphonic measures.



(a) Examples of measures with polyphonic parts of type 1.



(b) Examples of measures with polyphonic parts of type 2.

Figure 6.5: The two types of polyphony in measures.

Now that the measures are encoded, we can evaluate the clustering results. Two measures are considered to be rhythmically the same if their encoding is the same, meaning each tuple representing a note or rest in the first measure has exactly the same tuple in the other measure, and vice versa. Using this, we can evaluate the quality of our clustering results. Each unique rhythmic encoding found in a score is considered to be a separate class present in the score. We recorded the number of classes for each score in Table 6.1. Together with the measure assignments to the clusters, we can compute the homogeneity and completeness metric, and from that the V-measure, their harmonic mean, all of which we introduced in Section 2.4.2. We use the default $\beta = 1.0$ when computing V-measure, as we have no reason to value either homogeneity or completeness higher than the other. The metrics are computed for each score, the results are shown in Table 6.2. Knowing that these metrics are bounded between 0 and 1, with 1 being the most desirable clustering, we can see that the clustering results tend to be on the poor side. It is also clear that the clustering is relatively good

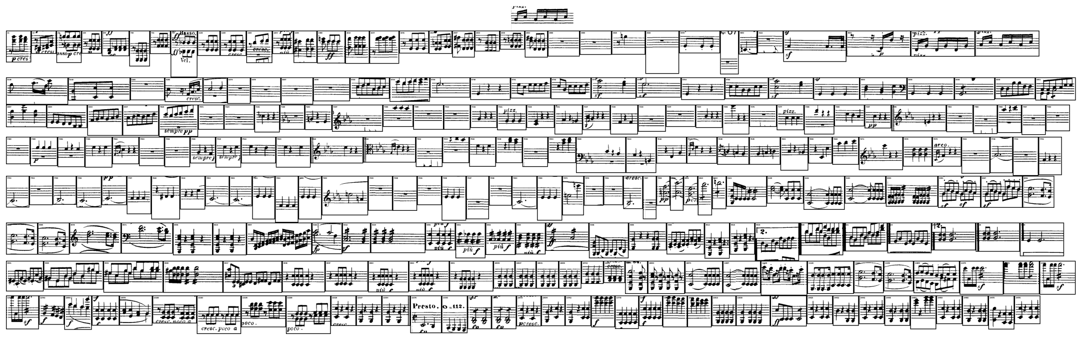


Figure 6.6: Cluster 18 of Beethoven's Symphony No. 5.

Table 6.1: The number of clusters and number of classes for each score in the dataset.

Score	Composer	#Clusters	#Classes
Brandenburg Concerto No. 5	Bach, J.S.	27	161
Symphony No. 1	Beethoven, L. van	27	260
Symphony No. 2	Beethoven, L. van	29	493
Symphony No. 3	Beethoven, L. van	26	417
Symphony No. 4	Beethoven, L. van	25	206
Symphony No. 5	Beethoven, L. van	25	352
Symphony No. 6	Beethoven, L. van	26	595
Symphony No. 7	Beethoven, L. van	27	460
Symphony No. 8	Beethoven, L. van	29	307
Symphony No. 9	Beethoven, L. van	31	1704
Symphony No. 3	Brahms, J.	28	740
Symphony No. 5	Bruckner, A.	22	578
Symphony No. 9	Bruckner, A.	23	474
The Planets	Holst, G.	24	989
Symphony No. 4	Mahler, G.	21	1336
Symphony No. 41	Mozart, W.A.	29	456
Ouverture 1812	Tchaikovsky, P.I.	24	471

in Bach Brandenburg Concerto No. 5 and Beethoven Symphony No. 4, which are the smallest scores in our dataset. The examples of clustering shown in Figure 6.3 were chosen because they are relatively small, and therefore serve as an easy way to visualize a clustering example, but the results of larger scores are generally not as good as those examples. An example of a cluster from a larger score can be seen in Figure 6.6, where a cluster of Beethoven's Symphony No. 5 is shown. We immediately see much more variation in what rhythmic classes are included in this cluster. We will discuss possible reasons and remedies for this in Chapter 7.

6.4.1. Investigating Usefulness of Clusters

Going back to the crowdsourcing setting, we could use these clusters to share the rhythmic information we have obtained from one measure to others: it would be much easier to determine the rhythm of a measure and identify which measures share its rhythm than it would be to determine the rhythm of each of these measures individually. To investigate whether our clustering results lend themselves for this purpose, we determine two things. First, we determine what groupings can be obtained from the clusters in which rhythmic information could be shared, and second, we determine whether the medoid is generally a good descriptor of its cluster, so that it could be used as a starting point to find these groupings. Ideally, the medoid would describe the largest rhythmic class contained in its cluster. If this is the case, we can leverage this in the crowdsourcing setting by grouping the measures with the same rhythm as the medoid instead of a random measure in the cluster, since the rhythmic class of that random measure could be an outlier in the current cluster.

The clusters we obtained are a good starting point to find measures with the same rhythm, but the rhythms are not perfectly separated. As we have seen in Table 6.1, the scores have far more rhythmic classes than clusters. Each cluster will contain multiple rhythmic classes and the usefulness of a cluster will depend on how much entropy there is within that cluster. The homogeneity metric gives some insight into this, as it is defined in terms of the entropy of the classes, conditional on the obtained clusters, see Equation 2.6. Even though the homogeneity results are on the low side, there is still the possibility that we can obtain groups of the same rhythmic class from these clusters in post-processing steps. In this crowdsourcing setting, for example, we could design a task in which users are asked to identify which measures have the same rhythm as a given measure.

Such a task would work best if we can give a measure as starting point, to which a user can compare the rhythms of other measures. The medoid would seem to be a logical choice for this, since it is the measure which has the smallest total deviation to all other measures in the cluster, and should therefore be the best descriptor of the cluster. To validate whether the medoid is indeed representative, we compute a medoid performance metric M_i for each cluster i. We do this by first grouping the measures of the cluster into the classes we obtained from the measure encodings. Then we check to which class the medoid belongs. M_i is then the size of the medoid's class in the cluster, normalized by the total cluster size. The larger the number of measures in the cluster that is accurately described by the medoid, the higher M_i will be, bounded by 0 and 1. The medoid performance metric for a score M is then the weighted average of all M_i of that score. Formally, given a cluster i, the class c to which the medoid of cluster i belongs, all measures assigned to cluster i A_i of length $|A_i|$, and all measures assigned to cluster i that belong to class c $A_{i,c}$ of length $\left|A_{i,c}\right|$, the medoid performance metric M for a score is defined as

$$M = \frac{1}{k}\sum_{i=1}^{k}\frac{\left|A_{i,c}\right|}{|A_i|} \tag{6.2}$$

with k again the number of selected clusters for a score.

The calculated values of M for each score are also included in Table 6.2. We see that its results are generally very low, which means that a relatively small part of the measures in a cluster is accurately represented by the medoid of that cluster. The low homogeneity values are a part of the explanation for this, but M is often far lower than the homogeneity metric. If the medoid always describes the largest rhythmic class within a cluster, we might see some sort of dependency between the homogeneity metric and M, but this is for many clusters not the case. This is supported when we look at the classes in which the medoid of each cluster is found. Only 159 of all 443 clusters in our dataset have its medoid in the largest rhythmic class in that cluster, in all other cases, there is a rhythmic class in that cluster that is larger than the one that contains the cluster medoid. This causes the medoid performance metric M to be so low and leads us to conclude that the medoid of these clusters is not a useful starting point for grouping measures together in the crowdsourcing setting.

Table 6.2: Homogeneity, completeness, V-measure, and medoid performance metric for the clusters generated from each of the scores.

Score	**Composer**	**Homogeneity**	**Completeness**	**V-measure**	***M***
Brandenburg Concerto No. 5	Bach, J.S.	0.553	0.652	0.599	0.495
Symphony No. 1	Beethoven, L. van	0.24	0.302	0.268	0.066
Symphony No. 2	Beethoven, L. van	0.227	0.335	0.27	0.061
Symphony No. 3	Beethoven, L. van	0.218	0.264	0.239	0.088
Symphony No. 4	Beethoven, L. van	0.483	0.517	0.5	0.527
Symphony No. 5	Beethoven, L. van	0.244	0.318	0.276	0.042
Symphony No. 6	Beethoven, L. van	0.247	0.374	0.297	0.021
Symphony No. 7	Beethoven, L. van	0.179	0.237	0.204	0.050
Symphony No. 8	Beethoven, L. van	0.197	0.241	0.216	0.114
Symphony No. 9	Beethoven, L. van	0.166	0.256	0.201	0.032
Symphony No. 3	Brahms, J.	0.297	0.465	0.362	0.176
Symphony No. 5	Bruckner, A.	0.327	0.435	0.373	0.308
Symphony No. 9	Bruckner, A.	0.305	0.421	0.354	0.212
The Planets	Holst, G.	0.275	0.373	0.316	0.234
Symphony No. 4	Mahler, G.	0.238	0.348	0.282	0.248
Symphony No. 41	Mozart, W.A.	0.353	0.47	0.403	0.214
Ouverture 1812	Tchaikovsky, P.I.	0.357	0.524	0.424	0.193

7
Discussion

In the previous chapters we have explained which steps we have taken, the methods we have used, and the results of our research. Now we are ready to discuss which parts went right and which parts could have gone better in this chapter. We will do this step by step, starting with the method of data acquisition in Section 7.1. Then we will evaluate our methods and results of the staff detection and barline detection steps in Section 7.2. Finally, we will discuss the methods and results of the measure clustering step in Section 7.3.

7.1. Data Acquisition

As we have shown in Chapter 3, we have set certain requirements to the scores we wanted to evaluate our work on. These requirements restricted us to work with scores for larger ensembles, instead of solo instruments or small groups, works from the 18th, 19th, and 20th century, and works that are typeset instead of handwritten. That last requirement followed in part from the first two because all of these works have been published as typeset documents. However, given the strict divide between research into handwritten scores and typeset scores in the OMR community, we felt it appropriate to make this requirement explicit. With these requirements, we first set out to obtain symbolic scores, as they are harder to obtain than printed ones. We did not end up using data from all available sources for two reasons. The first reason is the trustworthiness of the sources providing the symbolic scores. Not all sources can be trusted to provide accurate transcriptions, which would mean that the symbolic scores would need to be manually checked and corrected. An example of such a source is the MuseScore free sheet music library[1], where every user can submit their own transcriptions without guarantees of quality. In contrast, the OpenScore organisation only uses the MuseScore platform to make their transcriptions public, but assures each transcription is peer-reviewed, and so we deem this source trustworthy for our research. The second reason for not using a source is the inability of the `music21` toolkit to parse its data type. We decided not to use the LilyPond data that is available for example, since its format is not supported by the toolkit. Although this means that we were limited in our choice of data, we do not think that this had a negative influence on the diversity of the data. Our dataset is quite large, and we managed to include parts for choir and parts for single staff percussion instruments, as we mentioned specifically in Section 3.1, and therefore feel we have a sufficiently diverse dataset.

Performing the manual annotations was a very laborious process, and unfortunately, we cannot guarantee that the score mapping that we obtained from it is completely free of errors. However, we have described the checks that we have performed after the annotations were obtained and feel that these sufficiently minimise the risk of errors in our mapped dataset. It should only be possible for errors to occur if they occur in multiples, where a previous error is later cancelled out by a subsequent error, otherwise the checks we performed would have caught them. The most rigorous check would of course be to check each pair of mapped printed and symbolic measures, but since this is infeasible with the number of measures in this dataset, perhaps some further checks could be designed that decrease the likelihood of errors even further, while still keeping human input to a minimum.

[1]https://musescore.com/

7.2. Measure Detection

The measure detection step, which was a preliminary to the measure clustering step, has been divided into two components, with a chapter dedicated to each of them: Chapter 4 discussed staff detection and Chapter 5 discussed barline detection. In Section 6.1, we combined these two to find the measures in printed scores. Although it is not highlighted as such in this research, the measure detector itself could also have its own use cases outside of this work. A very early version of it has for example been used to segment a score into measures in a TROMPA study focused on task design for crowdsourced OMR, which can be found in Section 3 of the TROMPA final deliverable [21]. We will now discuss the staff detection and barline detection steps separately.

7.2.1. Staff Detection

In Chapter 4, we have taken a look at the performance of the Dalitz staff detection algorithm and evaluated it on the data we have collected. We quantified its performance by comparing the staffs found by the algorithm with the layout annotations that we have made in Chapter 3. The performance metric we used is the mean absolute error. As this led to quite good results, we have also analysed each of the errors and categorised them, so that it is easier to evaluate what the weaknesses of the algorithm are. As we already indicated in Section 4.1, we expected errors of category A, where staffs consisting of only a single staffline would not be detected properly. However, we did not expect there to be so many errors of category B, where staffs are not detected at all, or multiple staffs are detected as one single staff. This is likely due to the fact that the Dalitz staff detection algorithm will always try to find staffs of a set number of stafflines, and so it will either discard staffs for which some stafflines are not detected properly, or it will consolidate two partially detected staffs into a single staff.

With this evaluation as a benchmark on our data, we set out to adapt the Dalitz algorithm into our own algorithm. The main aim was to add support for staffs with only a single staffline, and to make it more robust to the other errors that we observed. We added single staffline support by removing the assumption that all staffs on a page have the same number of stafflines, and with that, the adapted algorithm should also be able to handle staffs consisting of other than 1 or 5 stafflines, although we have not evaluated this, since this does not occur very frequently in CWMN. We also added an additional final step which can rectify stafflines that were wrongfully joined with a staff or separated from a staff, because we observed that with our changes this could happen occasionally. We took the same approach to evaluate this algorithm as we did with the Dalitz algorithm. The results show that for all scores except one, our adaptation outperforms the Dalitz algorithm. The number of errors has been reduced significantly for each category, except for category D. This category, where horizontal elements are wrongfully detected as staffs, now has a larger chance of occurring than it did with the Dalitz algorithm, since these elements are sometimes confused with staffs consisting of one staffline. Errors of this category are also the reason that the Dalitz algorithm outperforms our adaptation in the Bruckner Symphony No. 9 score. Observing the results of our adaptation compared to the Dalitz algorithm, we can safely say that our adaption yields clear improvements, based on our dataset. The increase of errors in category D is a logical result of removing an important assumption of the Dalitz algorithm, but given that only a few additional errors are made, we think this is a tradeoff worth making. Furthermore, these errors could perhaps be reduced by trying to filter out these false positives. The algorithm could for example be modified to more be more strict towards staffs with only one staffline when removing all staffs that are too short. Another option would be to compare the distances in between all the detected staffs: if the distance from a staff with a single staffline to other staffs on the page presents an anomaly, perhaps it should be removed from the result.

7.2.2. Barline Detection

With staff detection done, we continued with barline detection in Chapter 5. Since the approach taken in the staff detection algorithm was so successful, we translated this as much as possible to barline detection. We obtained the segments in the same way, by running the skeletonization step of the staff detection algorithm on a rotated version of the input. After that, the segments were grouped into systems and later into barlines. Interpolation between segments of barline candidates helped to include as many segments as possible in the final result, and we did a final sweep where barlines are removed that are much shorter than the second-longest barline or have too small segments. We have evaluated the barline detection algorithm in the same manner as the staff detection algorithm, by comparing the number of found barlines to the layout annotations of Chapter 3 and computing a performance metric using the mean absolute error. Furthermore, the errors have again been analysed by dividing them into categories, which more accurately describe what mistakes the algorithm exactly makes.

This analysis makes it clear that most of the errors are due to missing barlines in the results. We expect that this is mainly due to step 8 of the algorithm, which removes barlines under certain circumstances. We have observed during testing that leaving out this step yields large amounts of false positives, where notes and rests in multiple staffs at approximately the same x coordinate would lead to a detected barline. Step 8 was designed to filter these false positives out, but it now also results in the removal of true positives. On the other hand, we still observe these false positives in error category B, and have observed a balance between the two when tweaking the criteria that determine which barlines should be removed. Finding the most ideal balance between these two will never yield a perfect solution, so to improve the results of the barline detection, we might have to look for alternative solutions in the algorithm. An example could be to increase the length threshold used in step 2 of the algorithm. The downside of this is that this reduces the number of segments the algorithm can work with. Systems with only a few staffs have small segments to begin with and could therefore suffer from this, especially when the barlines consist of only small barline segments per staff. In that case, all the candidate segments would be removed and the barline would still not be detected. An alternative solution might be to leverage the shape of the detected barlines. The barlines within a system will mostly consist of segments of similar lengths, and so when the segments of a barline deviate from this, it might be a good candidate for removal from the results. Furthermore, we have some errors in category D, where false positives are the result of noise generated by a scanner. Given that all these false positives are located outside the actual systems, it might be possible to use the horizontal boundaries of the detected staffs as horizontal system boundaries. Any barlines that fall outside these boundaries might have to be removed from the results. Although this will of course not prevent false positives from all scanner noise, this would most likely fix the barline detection errors in our dataset.

7.3. Measure Clustering

Finally, we covered measure clustering in Chapter 6. Given the staffs and barlines we detected before, we segmented each page into images of individual measures. The disputed space in between two vertically neighbouring measures was divided equally between the two, even though this might not separate the measures perfectly. We determined the distances between these measures through Dynamic Time Warping and used the k-medoids clustering algorithm to obtain clusters of measures. The number of clusters was determined by using the Kneedle algorithm to find the elbow in the k vs. TD plot for $k \in \{2 \ldots 150\}$. We evaluated these clusters using the homogeneity, completeness and V-measure metrics. Furthermore, we evaluated the usefulness of the clusters by discussing the possibility of obtaining groups of measures from the same rhythmic class, and quantifying how representative the medoids are for each cluster. There are several aspects of this process we would like to discuss, we do this below in order of occurrence in Chapter 6.

7.3.1. Combining Staffs and Barlines

First of all, combining the staffs and barlines was a straightforward process. We did however opt to remove the pages where the detected staffs and barlines did not match the obtained annotations, to avoid these from hampering our clustering evaluation. As mentioned, this removes over 10% of our data, but given that the errors are relatively spread out over all the scores in our dataset, as can be seen in Tables 4.1 and 5.1, this still leaves a dataset that is sufficiently large and diverse. Correcting these errors would require human input, but should not be a very hard task. Since both detection algorithms take clearly defined steps, a possible solution could be to identify which steps have the highest chance to result in wrong results, have the algorithm present its intermediate results after that step, require human input to fix any mistakes that the algorithm might have made up to that point, and then apply the remaining steps of the algorithms. Furthermore, we have used a very simple solution to determine the vertical limits of the bounding box of each measure. As can be seen in the example of Figure 6.1, this cannot always correctly separate the contents of the measures. Better linear separators could of course be devised, but none of them will yield perfect results. Regardless, we do not expect this to have negatively impacted the results very much, although we do not have the evidence to back this up. We expect that factors we will mention further down have had a much higher impact on the clustering performance than incorrect measure separation.

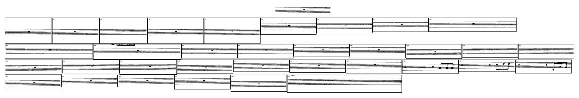
7.3.2. Distances between Measures

Using DTW as the distance metric turned out to be a suitable choice. It provided the support we needed for computing distances between time series of varying lengths, and the examples given in Figure 6.3 show that with this metric it is possible to relatively accurately depict the actual rhythmic distances. Something we chose not to do when using DTW is normalization of the time series. Generally, when using DTW, the value of each time point is z-normalized. We can afford to skip this step, however, since the typeset data gives us very stable symbols throughout the entire score. Each staffline, note stem, rest, etc. that represent the same rhythm will have the same dimensions in the score. Therefore, each difference in length between symbols is an important difference that needs to be represented in the data and normalization might actually hurt the distance calculation. The distance calculations did cost quite some time, even though we used the FastDTW algorithm, since there are simply very many time series to compare. Even using all 16 threads of an Intel Xeon E5-2560 processor, this took a full 24 hours for the Bruckner Symphony No. 9 score. Although this is not a problem in most settings, since distance computation is a one-time operation for an entire score, it is still a considerable amount of time and was the computational bottleneck of this thesis.

7.3.3. Evaluation of the Clusters

The k-medoids algorithm seems to be able to perform relatively well when looking at the examples in Figure 6.3. The fact that this is not reflected in the evaluation results is most likely due to the choice of k. We used the elbow method to find the point of diminishing returns, but in hindsight, this might not have been the best choice, though it depends on the use case. Increasing the number of clusters would most likely increase the homogeneity of each cluster, but this might in turn hurt the completeness. If the distances would perfectly describe the purely rhythmic distance, the completeness is also expected to increase when the number of clusters approaches the true number of classes, but since we work with time series generated from images, the distances are influenced by other factors as well. One such factor is the presence of other symbols than the notes and rests in a measure. Several symbols we did not include in evaluating the clusters are clefs, time signatures, and accidentals. These symbols have no direct effect on the rhythmic contents, but they do generate large values in the time series. This has a big effect on the clusters, which we illustrate with an example in Figure 7.1. The measures in both clusters have a very similar rhythm, apart from a few outliers, but we see in Figure 7.1a almost no measure with clefs and accidentals, and in Figure 7.1b almost exclusively measures with clefs and accidentals. The fact that these measures are separated into two different clusters means that the clefs and accidentals have had a large impact on clustering. This means that increasing the number of clusters would probably increase the homogeneity, but reduce the completeness of the clusters. Whether this tradeoff is worthwhile depends on the use case. In the crowdsourcing setting, this might actually be a tradeoff worth making. Increasing the number of clusters might make it easier to obtain groups of measures from the same rhythmic class, which can be used to make the transcription process more efficient. Increasing the number of clusters will also cause more measures that belong to the same rhythmic class to be spread out across more clusters, but this might not be a problem, since the groups that can be found already yield an improvement for the process. It does however mean that there is room for even more efficiency gains, since even larger groups could be formed if the clustering performed better.

Finally, the medoid performance metric M showed us that the medoids are not a good descriptor of the current clusters. Our hypothesis is that this is because there can be quite a few rhythmic classes in each cluster, and since the medoid minimizes the total deviation to all of them, the medoid balances between the rhythmic classes in the cluster, which renders the medoid useless. If the number of clusters is increased such that each cluster is dominated by a single rhythmic class, however, which is for example the case in Figure 6.3, then the medoid might become a good starting point when finding groups of the same rhythmic class.


(a) Cluster 6 of Bach Brandenburg Concerto No. 5.

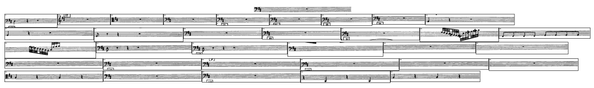

(b) Cluster 21 of Bach Brandenburg Concerto No. 5.

Figure 7.1: Two clusters of Bach's Brandenburg Concerto No. 5. The first contains no clefs or accidentals, while the second does almost exclusively.

8 Conclusion

In this final chapter, we will conclude our work. First, we will look back at the research questions and answer them in Section 8.1. Next, we will list our main contributions in Section 8.2, and finally, we will lay out some recommendations for future work in Section 8.3.

8.1. Answering the Research Questions

Now that all our methods have been explained and all our results are obtained, we circle back to the research questions introduced in Section 1.7 to answer them. We start with the subquestions:

***SQ1**: What data are needed to design and evaluate a digital system which can find similarities and how can we prepare the data for this task?*

We have identified the scope of music based on applicability to the crowdsourcing setting. We restricted ourselves to typeset scores from the 18th, 19th, and 20th century for larger ensembles, since those are representative of a very large pool of printed scores that could benefit from this research. We have collected both symbolic scores and printed scores and have designed a way to find a mapping between them at the measure level, so that we can evaluate the results of our digital system.

***SQ2**: How can we reliably detect the measures in a scanned music score?*

We have improved an existing staff detection algorithm to support staffs consisting of varying numbers of stafflines on a single page and reduce common errors. We have evaluated the adapted version of the algorithm against the original on our own dataset and concluded that it yields a large improvement and has a high success rate. Furthermore, we have developed a barline detection algorithm and evaluated that against our dataset as well. Although the barline detection algorithm makes more errors than the staff detection algorithm, the measures have been perfectly detected on 88.9% of the pages.

***SQ3**: How can we find similarities between detected measures based on rhythmic information?*

We have calculated the distance between each pair of measures in a score using Dynamic Time Warping and have used the FastPAM k-medoids algorithm to obtain clusters of measures. We determined the number of clusters by using the elbow method and have evaluated the clusters using the mapped symbolic scores as a ground truth. We reported the homogeneity, completeness, and V-measure metrics, together with a medoid performance evaluation metric M to make the case that the medoids in the current clusters are not suitable as a good descriptor of the clusters.

With the subquestions answered, we have arrived back at the main research question: *How can we design and evaluate a digital system that finds similarities in scanned music scores on the measure level?*. With the steps we have taken to answer the subquestions, we have answered all the individual parts of the main research question. These steps combined are themselves the final answer to the main research question.

8.2. Main Contributions

In this thesis, we have investigated how to build a system that can detect rhythmic similarities between measures of scanned music scores. The various parts of the system we have developed can each be used in their own sense, or can be used in combination with each other. We list our most important contributions here.

- We have provided a way of mapping symbolic scores to printed scores on the measure level with as little human input as possible.
- We have compiled an extensive dataset of large ensemble works from the 18th, 19th, and 20th century. Each work is available as both a symbolic score and printed score, and the mappings between the symbolic score and the printed score are also provided.
- We have developed an improvement to the Dalitz staff detection algorithm and provided a comparison between the two algorithms based on our dataset. We have also made the Dalitz algorithm available in Python 3, as the original implementation was only available in Python 2, which is now deprecated.
- We have developed a barline detection algorithm and evaluated it on our dataset.
- We have explored how measures could be grouped together by rhythmic similarity using a k-medoids clustering technique. We have evaluated the quality of these clusters, and have discussed how their results could be useful to a crowdsourcing setting.

The dataset[1] and implementations[2] are all made available on GitHub.

8.3. Recommendations for Future Work

Given our methods and findings, we have identified some possible directions for improvements and future research, which we will discuss here.

8.3.1. Extending the Scope

As we mentioned in Section 7.2.2, increasing the length threshold to improve barline detection might have a negative effect on systems with only one or two staffs. Since this does not occur frequently in works for larger ensembles, this case is not reflected in our dataset. However, including this type of music might be a good starting point when extending the scope of this research. Works written for solo instruments, of which there is a lot, could also be included in this extended scope. Although this might not all be useful in the similarity study, the staff and barline detection algorithms could prove to be useful within that scope.

Furthermore, an interesting extension could be to make a more granular mapping between the symbolic and printed scores. Right now, the mapping is done at the measure level, but a lot of OMR research is focused on symbol classification. There exist datasets for these tasks, but since most research effort in this task goes into machine learning or deep learning, the research effort could benefit from having more data available.

8.3.2. Measure Detection Recommendations

The staff detection algorithm we proposed already provides a significant improvement over the Dalitz staff detection algorithm, but it is not without flaws. As we have seen in Section 4.2, the most errors are made because long horizontal symbols are incorrectly detected as staffs. Even further improvements could be made to alleviate these errors, which would eliminate the last consistent source of errors from the algorithm. We discussed some possible solutions in Section 7.2.1.

We have seen in the previous chapters that the barline detection algorithm is not as robust as the staff detection algorithm, at least not on our dataset. We hinted at possible improvements in Section 7.2.2, which are to either find an alternative for step 8, where false positives are removed too aggressively, or perhaps avoid false positives to begin with by increasing the segment length threshold. Alternatively, other solutions to barline detection as a whole are of course still open for research.

Both the staff detection and barline detection algorithms do lend themselves quite well to error correction. Since they both have a clear set of steps they follow, error correction could be done by identifying which steps are the most likely to produce wrong results. Having a system where the intermediate results can be

[1] https://github.com/MathiasMeuleman/mapped-measures-dataset
[2] https://github.com/MathiasMeuleman/measure-clustering

checked and possibly corrected by a person, and only after that apply the remainder of the steps, could reduce the number of errors made. Post hoc error correction is of course also a possibility, by adding or removing staffs, stafflines, or barlines. Both setups lend themselves quite well for the crowdsourced setting.

8.3.3. Measure Clustering Recommendations

As we have seen in Section 6.4 and discussed in Section 7.3, clustering performance is something that can be improved upon. The chosen distance metric, Dynamic Time Warping, does not only compute rhythmic differences, but also takes other symbols into account, and so it makes sense that evaluating on rhythm alone will negatively impact clustering performance. This could be improved by either changing the evaluation method to take all other symbols into account as well, or by adapting the distance metric. The choice will have to depend on the use case.

In Section 6.4.1, we discussed how the clustering results could be useful in a crowdsourcing setting. The main challenge will be to obtain groups of measures that all have the same rhythm. A user task could be a possible solution for this, but it could also be possible that the number of clusters can be tweaked in such a way that the clusters are very likely to contain only one rhythmic class. This does of course come at the cost of having many clusters, but could be interesting for the crowdsourcing setting nonetheless. It could be that the medoids become better descriptors of the clusters, allowing them to be used as the first targets of the grouping task, if the approach of many clusters is chosen.

This page was intentionally left blank.

Bibliography

[1] David Bainbridge and Tim Bell. The challenge of optical music recognition. *Computers and the Humanities*, 35(2):95–121, May 2001. ISSN 1572-8412. doi: 10.1023/A:1002485918032. URL https://doi.org/10.1023/A:1002485918032.

[2] Pierfrancesco Bellini, Ivan Bruno, and Paolo Nesi. Optical music sheet segmentation. In *Proceedings First International Conference on WEB Delivering of Music. WEDELMUSIC 2001*, pages 183–190, 2001. doi: 10.1109/WDM.2001.990175.

[3] Pierfrancesco Bellini, Ivan Bruno, and Paolo Nesi. Assessing optical music recognition tools. *Computer Music Journal*, 31(1):68–93, 2007.

[4] Richard Bellman and Robert Kalaba. On adaptive control processes. *IRE Transactions on Automatic Control*, 4(2):1–9, 1959. ISSN 1558-3651. doi: 10.1109/TAC.1959.1104847. URL https://doi.org/10.1109/TAC.1959.1104847.

[5] Manuel Burghardt and Sebastian Spanner. Allegro: User-centered design of a tool for the crowdsourced transcription of handwritten music scores. In *Proceedings of the 2nd International Conference on Digital Access to Textual Cultural Heritage*, DATeCH2017, page 15–20, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450352659. doi: 10.1145/3078081.3078101. URL https://doi.org/10.1145/3078081.3078101.

[6] Donald Byrd and Jakob Grue Simonsen. Towards a standard testbed for optical music recognition: Definitions, metrics, and page images. *Journal of New Music Research*, 44(3):169–195, 2015. doi: 10.1080/09298215.2015.1045424. URL https://doi.org/10.1080/09298215.2015.1045424.

[7] Jorge Calvo-Zaragoza, Jose J. Valero-Mas, and Antonio Pertusa. End-to-end optical music recognition using neural networks. In *Proceedings of the 18th International Society for Music Information Retrieval Conference*, page 472–477. ISMIR, Oct 2017. ISBN 978-981-11-5179-8. doi: 10.5281/zenodo.1418333.

[8] Liang Chen and Christopher Raphael. Human-directed optical music recognition. *Electronic Imaging*, 2016(17):1–9, 2016. ISSN 2470-1173. doi: 10.2352/ISSN.2470-1173.2016.17.DRR-053. URL https://www.ingentaconnect.com/content/ist/ei/2016/00002016/00000017/art00003.

[9] Liang Chen, Erik Stolterman, and Christopher Raphael. Human-interactive optical music recognition. In *Proceedings of the 17th International Society for Music Information Retrieval Conference*, page 647–653. ISMIR, Aug 2016. ISBN 978-0-692-75506-8. doi: 10.5281/zenodo.1416184.

[10] Christoph Dalitz, Michael Droettboom, Bastian Pranzas, and Ichiro Fujinaga. A comparative study of staff removal algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(5):753–766, 2008. doi: 10.1109/TPAMI.2007.70749.

[11] David L. Davies and Donald W. Bouldin. A cluster separation measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-1(2):224–227, 1979. ISSN 1939-3539. doi: 10.1109/TPAMI.1979.4766909. URL https://doi.org/10.1109/TPAMI.1979.4766909.

[12] J. Stephen Downie, Kris West, Andreas Ehmann, and Emmanuel Vincent. The 2005 music information retrieval evaluation exchange (mirex 2005): preliminary overview. In *6th Int. Conf. on Music Information Retrieval (ISMIR)*, pages 320–323, London, United Kingdom, Sep 2005. URL https://hal.inria.fr/inria-00544675.

[13] Michael Droettboom, Karl Macmillan, and Ichiro Fujinaga. The gamera framework for building custom recognition systems. In *Proceedings - 2003 Symposium on Document Image Understanding Technology*, pages 275–286, Apr 2003.

[14] Ichiro Fujinaga. *Adaptive Optical Music Recognition*. PhD thesis, McGill University, June 1996.

[15] Michael Good. MusicXML: An internet-friendly format for sheet music. In *Xml conference and expo*, pages 03–04. Citeseer, 2001.

[16] IMSLP. Main webpage. https://imslp.org/wiki/Main_Page, 2021. [Online; accessed May 3 2021].

[17] Fumitada Itakura. Minimum prediction residual principle applied to speech recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 23(1):67–72, 1975. ISSN 0096-3518. doi: 10.1109/TASSP.1975.1162641. URL https://doi.org/10.1109/TASSP.1975.1162641.

[18] Leonard Kaufman and Peter J Rousseeuw. Clustering by means of medoids. In *Proceedings of the Statistical Data Analysis Based on the L1 Norm Conference, Neuchatel, Switzerland*, pages 405–416, 1987.

[19] Leonard Kaufman and Peter J Rousseeuw. *Finding groups in data: an introduction to cluster analysis*, volume 344. John Wiley & Sons, 2009.

[20] Joseph B. Kruskal and Mark Liberman. The symmetric time-warping problem: From continuous to discrete. *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*, pages 125–161, 01 1983.

[21] Cynthia Liem, David Baker, Ioannis Samiotis, David Weigl, Nicolás Gutiérrez, Juan Gómez, Maria Pascual, and Emilia Gómez. Deliverable 6.9 - final evaluation, 2021. URL https://trompamusic.eu/deliverables/TR-D6.9-Final_Evaluation.pdf.

[22] Karl MacMillan, Michael Droettboom, and Ichiro Fujinaga. Gamera: Optical music recognition in a new shell. In *Proceedings of the 2002 International Computer Music Conference, ICMC 2002, Gothenburg, Sweden, September 16-21, 2002*. Michigan Publishing, 2002. URL http://hdl.handle.net/2027/spo.bbp2372.2002.098.

[23] James MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA, 1967.

[24] Hidetoshi Miyao. Stave extraction for printed music scores. In Hujun Yin, Nigel Allinson, Richard Freeman, John Keane, and Simon Hubbard, editors, *Intelligent Data Engineering and Automated Learning — IDEAL 2002*, pages 562–568, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg. ISBN 978-3-540-45675-9. doi: 10.1007/3-540-45675-9_85.

[25] Marcel Mongeau and David Sankoff. Comparison of musical sequences. *Computers and the Humanities*, 24(3):161–175, Jun 1990. ISSN 1572-8412. doi: 10.1007/BF00117340. URL https://doi.org/10.1007/BF00117340.

[26] Daniel Müllensiefen and Klaus Frieler. Evaluating different approaches to measuring the similarity of melodies. In Vladimir Batagelj, Hans-Hermann Bock, Anuška Ferligoj, and Aleš Žiberna, editors, *Data Science and Classification*, pages 299–306, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. ISBN 978-3-540-34416-2. doi: 10.1007/3-540-34416-0_32.

[27] Raymond T. Ng and Jiawei Han. CLARANS: a method for clustering objects for spatial data mining. *IEEE Transactions on Knowledge and Data Engineering*, 14(5):1003–1016, Sep 2002. ISSN 1558-2191. doi: 10.1109/TKDE.2002.1033770. URL https://doi.org/10.1109/TKDE.2002.1033770.

[28] Vit Niennattrakul and Chotirat Ann Ratanamahatana. Clustering multimedia data using time series. In *2006 International Conference on Hybrid Information Technology*, volume 1, pages 372–379, 2006. doi: 10.1109/ICHIT.2006.253514.

[29] Vit Niennattrakul and Chotirat Ann Ratanamahatana. On clustering multimedia time series data using k-means and dynamic time warping. In *2007 International Conference on Multimedia and Ubiquitous Engineering (MUE'07)*, pages 733–738, 2007. doi: 10.1109/MUE.2007.165.

[30] Alexander Pacha, Jorge Calvo-Zaragoza, and Jan Hajič jr. Learning notation graph construction for full-pipeline optical music recognition. In *Proceedings of the 20th International Society for Music Information Retrieval Conference*, page 75–82. ISMIR, Nov 2019. doi: 10.5281/zenodo.3527744. URL https://zenodo.org/record/3527744.

[31] Dennis H. Pruslin. Automatic recognition of sheet music, 1966. a critical survey of music image analysis. *Structured document image analysis. Springer, Heidelberg*, pages 405–434, 1992.

[32] Ana Rebelo, G Capela, and Jaime S Cardoso. Optical recognition of music symbols. *International Journal on Document Analysis and Recognition (IJDAR)*, 13(1):19–31, Mar 2010. ISSN 1433-2825. doi: 10.1007/s10032-009-0100-1. URL https://doi.org/10.1007/s10032-009-0100-1.

[33] Ana Rebelo, Ichiro Fujinaga, Filipe Paszkiewicz, Andre R. S. Marcal, Carlos Guedes, and Jaime S. Cardoso. Optical music recognition: state-of-the-art and open issues. *International Journal of Multimedia Information Retrieval*, 1(3):173–190, Oct 2012. ISSN 2192-662X. doi: 10.1007/s13735-012-0004-6. URL https://doi.org/10.1007/s13735-012-0004-6.

[34] John W. Roach and J.E. Tatem. Using domain knowledge in low-level visual processing to interpret handwritten music: An experiment. *Pattern Recognition*, 21(1):33–44, 1988. ISSN 0031-3203. doi: 10.1016/0031-3203(88)90069-6. URL https://www.sciencedirect.com/science/article/pii/0031320388900696.

[35] Perry Roland. The Music Encoding Initiative (MEI). In *Proceedings of the First International Conference on Musical Application Using XML*, volume 1060, pages 55–59, 2002.

[36] Andrew Rosenberg and Julia Hirschberg. V-measure: A conditional entropy-based external cluster evaluation measure. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 410–420, Prague, Czech Republic, Jun 2007. Association for Computational Linguistics. URL https://aclanthology.org/D07-1043.

[37] Peter J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, 1987. ISSN 0377-0427. doi: 10.1016/0377-0427(87)90125-7. URL https://www.sciencedirect.com/science/article/pii/0377042787901257.

[38] Hiroaki Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 26(1):43–49, 1978. ISSN 00963518. doi: 10.1109/TASSP.1978.1163055.

[39] Stan Salvador and Philip Chan. Determining the number of clusters/segments in hierarchical clustering/segmentation algorithms. In *Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence*, pages 576–584, 2004. doi: 10.1109/ICTAI.2004.50. URL https://doi.org/10.1109/ICTAI.2004.50.

[40] Stan Salvador and Philip Chan. Toward accurate dynamic time warping in linear time and space. *Intelligent Data Analysis*, 11:561–580, 2007. ISSN 1571-4128. doi: 10.3233/IDA-2007-11508. URL https://doi.org/10.3233/IDA-2007-11508. 5.

[41] Ville Satopää, Jeannie Albrecht, David Irwin, and Barath Raghavan. Finding a "kneedle" in a haystack: Detecting knee points in system behavior. In *Proceedings of the 31st International Conference on Distributed Computing Systems Workshops*, 2011 31st International Conference on Distributed Computing Systems Workshops, pages 166–171, 2011. doi: 10.1109/ICDCSW.2011.20. URL https://doi.org/10.1109/ICDCSW.2011.20.

[42] Erich Schubert and Peter J. Rousseeuw. Fast and eager k-medoids clustering: O(k) runtime improvement of the PAM, CLARA, and CLARANS algorithms. *Information Systems*, 101, 2021. ISSN 0306-4379. doi: 10.1016/J.IS.2021.101804. URL https://www.sciencedirect.com/science/article/pii/S0306437921000557.

[43] Claude E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3): 379–423, 1948. ISSN 0005-8580. doi: 10.1002/j.1538-7305.1948.tb01338.x. URL https://doi.org/10.1002/j.1538-7305.1948.tb01338.x.

[44] Mariusz Szwoch. A robust detector for distorted music staves. In André Gagalowicz and Wilfried Philips, editors, *Computer Analysis of Images and Patterns*, pages 701–708, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. ISBN 978-3-540-32011-1. doi: 10.1007/11556121_86.

[45] The MIDI Association. Official midi specifications. https://www.midi.org/specifications, 2021. [Online; accessed November 15 2021].

[46] TROMPA. About webpage. https://trompamusic.eu/about-trompa, 2021. [Online; accessed May 3 2021].

[47] Lukas Tuggener, Ismail Elezi, Jürgen Schmidhuber, Marcello Pelillo, and Thilo Stadelmann. Deepscores-a dataset for segmentation, detection and classification of tiny objects. In *Proceedings of the 24th International Conference on Pattern Recognition (ICPR)*, pages 3704–3709, 2018. doi: 10.1109/ICPR.2018.8545307. URL https://doi.org/10.1109/ICPR.2018.8545307.

[48] Eelco van der Wel and Karen Ullrich. Optical music recognition with convolutional sequence-to-sequence models. In *Proceedings of the 18th International Society for Music Information Retrieval Conference*, page 731–737. ISMIR, Oct 2017. ISBN 978-981-11-5179-8. doi: 10.5281/zenodo.1415664.

[49] Valerio Velardo, Mauro Vallati, and Steven Jan. Symbolic Melodic Similarity: State of the Art and Future Challenges. *Computer Music Journal*, 40(2):70–83, 06 2016. ISSN 0148-9267. doi: 10.1162/COMJ_a_00359. URL https://doi.org/10.1162/COMJ_a_00359.

[50] Gabriel Vigliensoni, Gregory Burlet, and Ichiro Fujinaga. Optical measure recognition in common music notation. In *Proceedings of the 14th International Society for Music Information Retrieval Conference*, page 125–130. ISMIR, Nov 2013. ISBN 978-0-615-90065-0. doi: 10.5281/zenodo.1417024.

[51] Simon Waloschek, Aristotelis Hadjakos, and Alexander Pacha. Identification and cross-document alignment of measures in music score images. In *Proceedings of the 20th International Society for Music Information Retrieval Conference*, page 137–143. ISMIR, Nov 2019. doi: 10.5281/zenodo.3527760.

[52] Frank Zalkow, Angel Villar Corrales, T J Tsai, Vlora Arifi-Müller, and Meinard Müller. Tools for semi-automatic bounding box annotation of musical measures in sheet music. In *20th Conference of ISMIR - Late Breaking/Demo Session*, 2019. URL https://www.audiolabs-erlangen.de/resources/MIR/2019-ISMIR-LBD-Measures.