GRAPH-BASED RHYTHM INTERPRETATION IN OPTICAL MUSIC RECOGNITION

Rong Jin

Submitted to the faculty of the University Graduate School

in partial fulfillment of the requirements

for the degree

Doctor of Philosophy in the School of Informatics, Computer Science, and Engineering,

Indiana University

November 2017

Accepted by the Graduate Faculty, Indiana University, in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Doctoral Committee

_____

Christopher Raphael, PhD

_____

David Crandall, PhD

_____

Julian Hook, PhD

_____

Daniel McDonald, PhD

Date of Defense: 08/22/2017

To my husband, Simo Zhang
My parents, Qing Min Jin and Jian Sun
And
My friends

Acknowledgements

First I thank my supervisor, Christopher Raphael, for his guidance over the past years. There will be no thesis without his help. He taught me to be a focused, persistent, patient and independent person.

I would also like to thank my doctoral committee members: David Crandall, Julian Hook and Daniel McDonald. The final project in David's computer vision class is one of the reasons that lead me to work on optical music recognition as my thesis topic. I would also like to thank Zhiyao Duan and Ichiro Fujinaga for introducing me to the field of music informatics when I was an undergraduate at Tsinghua University.

My previous and current lab mates, Yushen Han, Eric Nichols, Yupeng Gu, Liang Chen, Yucong Jiang, and Sonna Wager also gave me generous help and support.

Lastly and most importantly, I thank my family and friends for being by my side. My husband, Simo Zhang, gives me the biggest strength to overcome all the difficulties and challenges. My parents gave me all the support and love they can provide. I am so grateful to have my friends, Guo Zhang, Chunyang Zhang, Di Wu, Jing Yang, Wen Li, Grace Law and Can Liu. Thank you for being in my journey and letting me be in yours.

Rong Jin

GRAPH-BASED RHYTHM INTERPRETATION IN OPTICAL MUSIC RECOGNITION

Optical Music Recognition (OMR) is a process that automatically converts the image of a music score into symbolic data. OMR can be divided into two main steps: recognition, the goal of which is to recognize "valid" music symbols, and interpretation, to understand the music meaning, such as pitch and rhythm. We focus on the interpretation problem, and more specifically, rhythm interpretation on piano scores.

In this thesis work, we propose a graph-based algorithm, which interprets rhythm by building a *rhythm graph* on all symbols in a system measure. Our approach represents the notes and rests in a system measure as the vertices of a graph. Then we build the graph by adding *voice edges* and *coincidence edges* between pairs of vertices. The graph is constructed under the constraint such that it leads to a meaningful rhythm interpretation. We score the graph based on music notation rules and choose the graph that has the best score. The problem is thus converted into a constrained optimization problem of finding the graph with the highest score. The rhythmic interpretation follows simply from the connected rhythm graph.

To evaluate the graph-based algorithm, we perform an experiment on a dataset specifically built to cover different types of rhythmic challenges encountered in polyphonic piano scores. We conclude that our algorithm is capable of applying measure level notation rules and finding the globally optimal interpretation, even in examples with splitting and merging voices as well as missing tuplets.

_____

Christopher Raphael, PhD

_____

David Crandall, PhD

_____

Julian Hook, PhD

_____

Daniel McDonald, PhD

Contents

List of figures

List of tables

1 Introduction

Symbolic music data is a digital data type that stores music information written on the score (including instruments, pitches, rhythms, dynamics, lyrics, etc.) in a structured data file such as MusicXML, MEI, etc. Symbolic music data has a wide range of applications. They can be imported into score writing programs such as Finale or Sibelius, so one can transpose and convert scores to parts easily. They can also be converted to MIDI files and played back. With a large database of symbolic music data, one can also search music by melodies, rhythms and lyrics or apply data mining to do musical analysis on different genres.

However, it is not easy to produce symbolic music data. A very common method for music symbolic data entry combines synthesizer keyboard entry and computer keyboard entry. The synthesizer keyboard is used to enter the notes by playing each voice separately. The computer keyboard is used to correct mistakes and enter other notations such as lyrics, slurs, and dynamics [1]. Such a process requires a skillful computer/keyboard user as well as time-consuming editing. As an alternative to manual input, Optical Music Recognition (OMR) is a process that converts images of music scores into symbolic data automatically.

OMR can be divided into two main steps: recognition, the goal of which is to recognize “valid” music symbols, and interpretation, to understand the music’s meaning, for instance to interpret the pitch and rhythm of the music symbols [2]. Most OMR research literature has been focused on the recognition step. On the other hand,

interpretation, although less studied in the OMR literature, is a challenging problem. Interpretation can be further divided into sub problems of pitch interpretation, rhythm interpretation, etc. In this work, we make rhythm interpretation in OMR as our focus. In the following sections, we show what rhythm interpretation is and why rhythm interpretation is a challenging problem.

1.1 A brief history of rhythm notation

Rhythm notation was far from precise in its early history. When music notation was first used to record single-voice singing tones, notes were usually sung freely in rhythm. *Neume* notation is the basic element of Western and Eastern music notation since the 6th century. The earliest form of *Neumes* only contained two signs, *Acutus* ( ), for rising inflection and *Gravis* ( ), for falling inflection. They were apparently not capable of conveying information of exact pitch or duration. In the 6th to 8th century, *Acutus* and *Gravid* were gradually replaced by *virga* ( ) and *punctum* ( ), representing similar meanings. And between the 9th and 13th century, the original *virga* and *punctum*s came to be written as a squared note form as ( ) and ( ). This evolution was primarily the result of their being placed on staff lines, which was a major development for representing *exact* pitch in music notation. However, a single Neume note still does not convey duration information by itself.

Duration on a single note did not appear until *Mensural* notation. The biggest innovation of *Mensural* notation is the systematic use of note shapes to denote rhythmic durations. In the 13th century, music expanded from a single voice to two or multiple independent parts. Two or multiple voices need to coordinate in time, which requires

more strict duration control methods. At this time, staff lines came to locate the exact pitch of a note, hence note shapes did not need to carry pitch information anymore. The two forms of the single-note Neume notes, *virga* ( ) and *punctum* ( ) became the symbols used for long and short time duration. They are pronounced as *longa* and *brevis* in their rhythmic form. This is the first time that the shape of a single note carried duration information. In the 14th century, *Semi-brevis* ( ) and *Minima* ( ) were used to represent half the duration of their higher-level note. In 15th century, more note shapes including *Semi-minima*, *Fusa* and *Semi-fusa* are included to represent more hierarchical durations. Coloration on note head (black and red) was introduced to express the different numerical duration relations under different meters (ternary or binary). Later, black and white note heads became standard and had been extended to the modern western music notation. At this time, Mensural notation closely corresponded the modern music notation.

The evolution of notes can be followed in Figure 1.1. The figure is originally from [3].

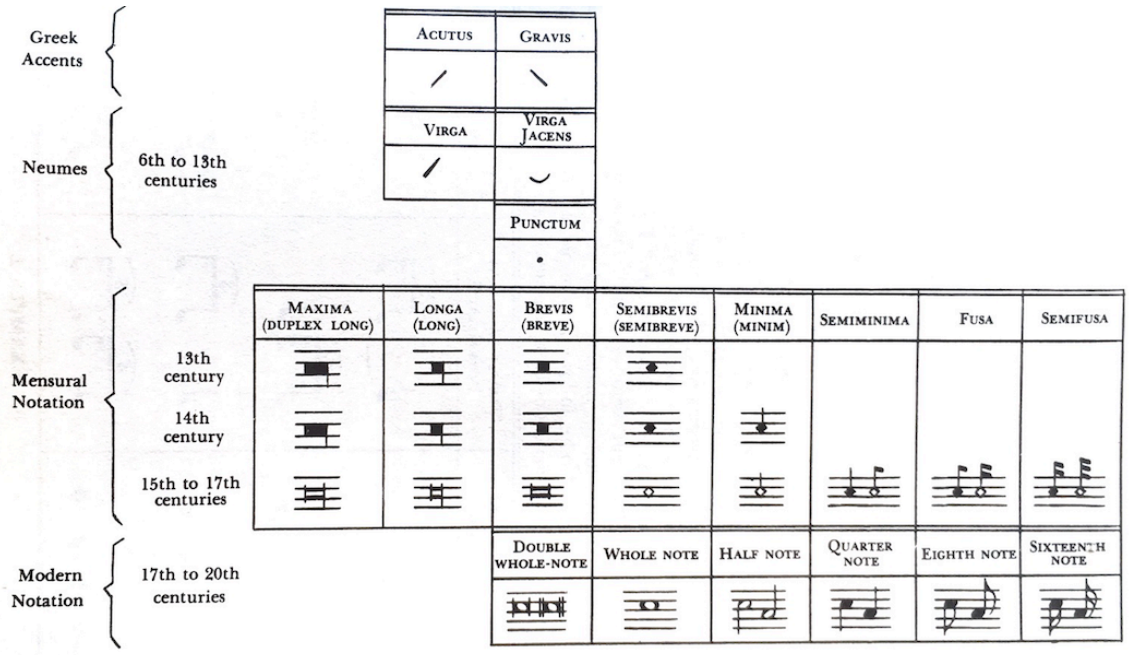


Figure 1.1: Development of note types in historic notations, from [3].

1.2 Rhythm interpretation in modern notation

In this section, we come back to the modern Western music notation. Music notation primarily carries two types of information, pitch and rhythm. Since our topic is rhythm interpretation in OMR, we will only introduce the notations that concern rhythm, which include definitions of measures, time signature, and durations of notes and rests. Then we will give two examples to demonstrate how to interpret rhythm in monophonic and polyphonic music.

1.2.1 Measure and time signature

A measure is commonly used to segment musical time into a specific number of beats. Measures are partitioned by barlines. Most music pieces have a repeated pattern of strong and weak beats. A measure segments these patterns accordingly. For example, a

music piece that has a pattern of 1-2-3-4-1-2-3-4 is divided into measures with four beats, usually with the strong beat as the first beat of the measure.

Besides emphasizing the strong beat, measures play additional rules. In music with multiple voice parts, measures also serve the duty of coordinating multiple parts. In piano or orchestra scores, bar lines span across multiple staves and form the structure of *system measures*.

In metered music, the length of a measure is defined by the time signature. The time signature is usually written at the beginning of a piece as a rational number such as $2/4$ or $3/4$. The denominator represents the basic unit of the beats, while the numerator represents the number of basic units in a measure. For example, $4/4$ means there are four beats in a measure, and each beat has the length of a quarter note.

Measures are an important unit in rhythm notation. We interpret rhythm in terms of musical time relative to the start of a measure. We define a measure to start at time 0 and end on time T, the time signature.

1.2.2 Duration and tuplet duration

In most modern Western music notation, the duration of a symbol is represented as a rational number. The duration of a note depends on its shape, which includes the type of the note head, stem, number of beams or flags and the type of augmentation dots. The duration of a rest also depends on its shape and the type of augmentation dots. A whole note is defined to have the unit duration of 1. Other notes have relative durations according to ratio relations, i.e. a half note has a duration of $1/2$ and an eighth note has

a duration of 1/8. Figure 1.2 shows a table of notes and rests with different durations and their relations.

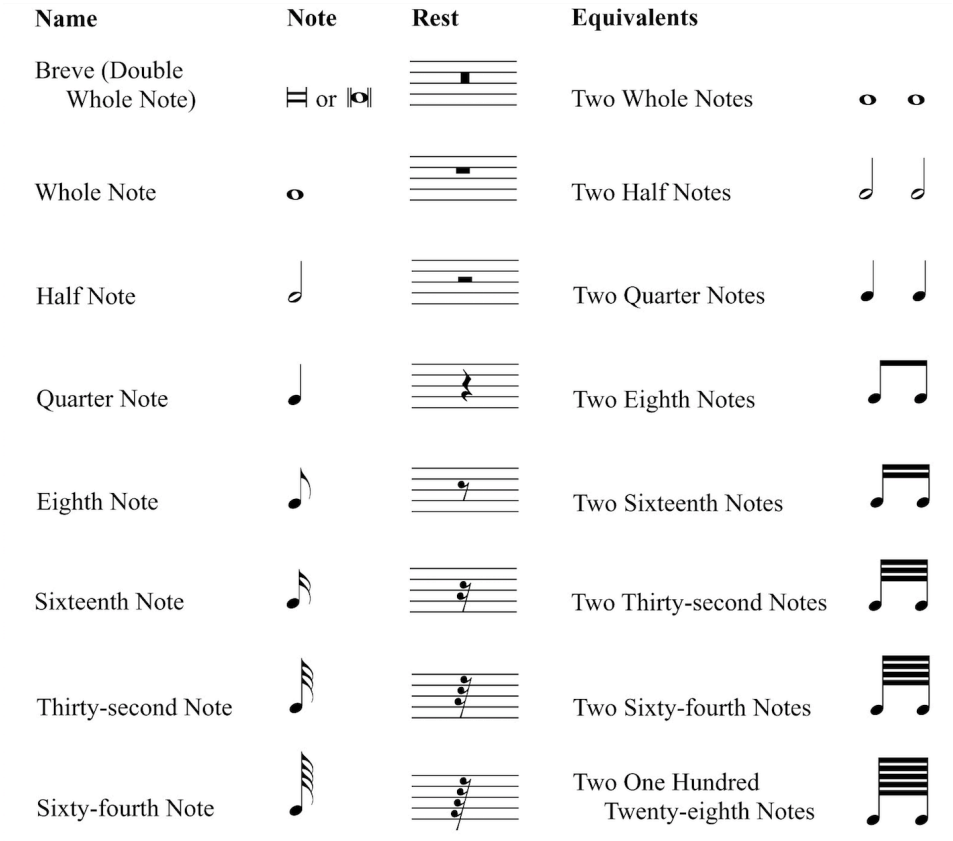


Figure 1.2: Different note durations in modern Western music notation.

The duration we just introduced is also called *nominal* duration, which only depends on the symbolic representation of the note. Just as in English, a word can have different meanings in different contexts; so too in music, a single symbol could have a different duration, a *tuplet* duration.

A *tuplet* is any rhythm that divides the beat into a different number of equal subdivisions from that usually implied by the time signature. For example, a 3-tuplet (or triplet) eighth note has duration of 1/12 instead of 1/8. A tuplet label is usually

written as a number or a ratio with a bracket on a group of symbols, as shown in Figure 1.3.

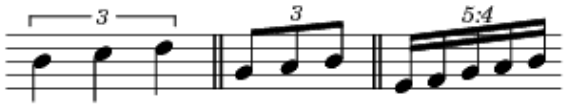


Figure 1.3: A standard triplet, a triplet denoted without a bracket, and a tuplet denoted as a ratio.

A *triplet* is by far the most common tuplet. When a triplet appears in a piece of music, it usually appears repeatedly. Therefore triplet labels are often only labeled in the first measure, and are omitted afterward to avoid crowding the score in the following measures, as shown in Figure 1.4. We call the symbols that are truly triplet but not explicitly labeled on the score "missing triplets". Thus, to interpret the correct rhythm, one needs to identify which symbols are missing triplets.

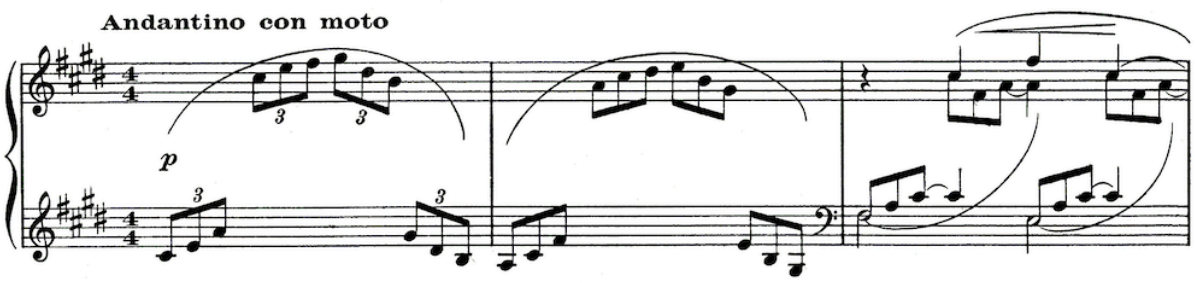


Figure 1.4: An example of missing triplets in the second and the third measures.

1.2.3 Monophony and polyphony

In order to play back music, we not only need the duration but also when each note is played. So we define *onset* as the musical time that a note is played relative to the start of a measure, where the onset is also represented as a rational number. With both

duration and onset of each symbol, we are able to represent rhythm. Obtaining onsets takes a little more work. We will show two examples, one in *monophonic* (one voice) and the other in *polyphonic* (several voices) music.

The first example is a monophonic measure, as shown in Figure 1.5. *Monophonic* music is the type of music score where there is only a single line of independent melody or one *voice* in the measure. The symbols in monophonic music are played one by one from left to right. We can sort all symbols in order and connect the neighboring symbols with a blue line, which we define as *voice edge*. The relation between the two consecutive symbols, v_i and v_{i+1}, connected by a voice edge, can be represented as:

$$x^o_{v_{i+1}} = x^o_{v_i} + x^d_{v_i} \tag{1.1}$$

where x^o_v and x^d_v denotes the onset and duration of symbol v. That is, the onset of a symbol v_{i+1} is equal to the sum of the onset and duration of its *predecessor*, symbol v_i. We also define the left bar line to have an onset of 0, and the right bar line to have an onset of T, the time signature. Since all symbols are connected by voice edges, we can represent the rhythm of the measure as a list of (onset, duration) pairs:

$$\left(\frac{0}{1},\frac{1}{16}\right),\left(\frac{1}{16},\frac{1}{16}\right),\left(\frac{1}{8},\frac{1}{16}\right),\left(\frac{3}{16},\frac{1}{32}\right),\left(\frac{7}{32},\frac{1}{32}\right),\left(\frac{1}{4},\frac{1}{16}\right),\left(\frac{5}{16},\frac{1}{16}\right),\left(\frac{3}{8},\frac{1}{8}\right),\ldots$$

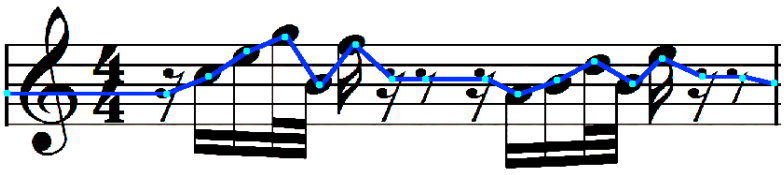


Figure 1.5: A measure showing an example of monophonic music. Neighboring symbols are connected to a sequence graph by voice edges (blue lines).

The second example is a polyphonic measure, as shown in Figure 1.6. Polyphonic measures refer to measures where there are at least two voices. The measure in Figure 1.6 has two voices, and we color them black and red. After separating into two voices, symbols are played in order within their own voices. We can connect neighboring symbols within the same voice with voice edges (blue lines). Equation 1.1 can now be written as

$$x_{v_j}^o = x_{v_i}^o + x_{v_i}^d \tag{1.2}$$

where symbol v_i and v_j are consecutive symbols in the same voice.

Similarly, once all symbols are connected to a connected graph, we get the rhythm interpretation for both voices according to equation 1.2:

Rhythm of the black notes: $\left(\frac{0}{1}, \frac{1}{8}\right), \left(\frac{1}{8}, \frac{1}{4}\right), \left(\frac{3}{8}, \frac{1}{4}\right), \left(\frac{5}{8}, \frac{1}{8}\right)$

Rhythm of the red notes: $\left(\frac{0}{1}, \frac{1}{4}\right), \left(\frac{1}{4}, \frac{3}{16}\right), \left(\frac{7}{16}, \frac{1}{16}\right), \left(\frac{1}{2}, \frac{3}{16}\right), \left(\frac{11}{16}, \frac{1}{16}\right)$

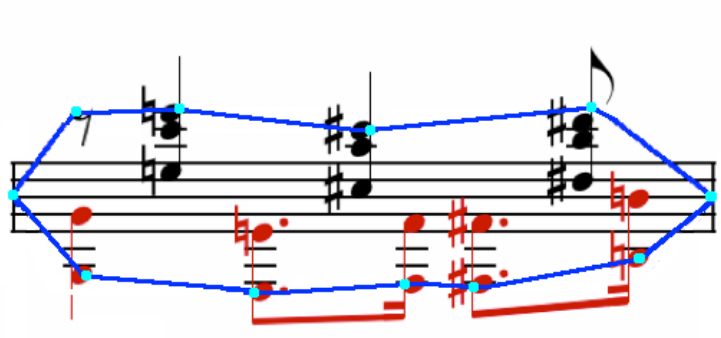


Figure 1.6: An example of polyphonic music where the two voices are colored in black and red. Symbols are connected into a connected graph by voice edges.

In the example in Figure 1.6, two complete voices stay all the way throughout a measure. However, sometimes voices can split and merge. Figure 1.7 shows a typical example where the number of voices changes from one to two and back to one. The eighth rest and quarter rest do double duties in both voices.

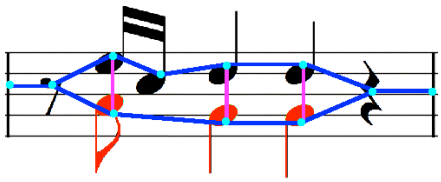


Figure 1.7: An example of voice splitting and merging. The time signature is in 4/4.

One way to interpret onsets is through voice edges, as shown in Figure 1.7. Another way is through the *vertical alignment* rule. This rule states that if two symbols align vertically, they usually have the same onset time. Therefore we use a pink line, which we denote as *coincidence edge*, to connect notes that they align. The onsets between two symbols, v_i and v_j connected by a coincidence edge, can be expressed as,

$$x_{v_j}^o = x_{v_i}^o \tag{1.3}$$

where x_v^o denotes the onset of symbol v. That is, two notes connected by a coincidence edge have the same onset time. The two types of edges lead to the same rhythm interpretation as follows.

Rhythm of the black notes: $\left(\frac{0}{1},\frac{1}{8}\right),\left(\frac{1}{8},\frac{1}{16}\right),\left(\frac{3}{16},\frac{1}{16}\right),\left(\frac{1}{4},\frac{1}{4}\right),\left(\frac{1}{2},\frac{1}{4}\right),\left(\frac{3}{4},\frac{1}{4}\right)$

Rhythm of the red notes: $\left(\frac{1}{8},\frac{1}{8}\right),\left(\frac{1}{4},\frac{1}{4}\right),\left(\frac{1}{2},\frac{1}{4}\right)$

1.3 Challenges of rhythm interpretation

The previous examples are relatively straightforward. So what can go wrong in rhythm interpretation? The answer can be shown through several examples on SmartScore [4]. SmartScore is one of the leading commercial OMR systems on the market. Like other OMR systems, SmartScore takes a scanned score image as input, recognizes symbols, and interprets the music meaning of each measure. When it shows music symbolic results, it checks if the result is musically meaningful. If it finds contradictions (usually rhythm contradictions), it flags the measure (as shown in Figure 1.8) and suggests to the user that something could be wrong.

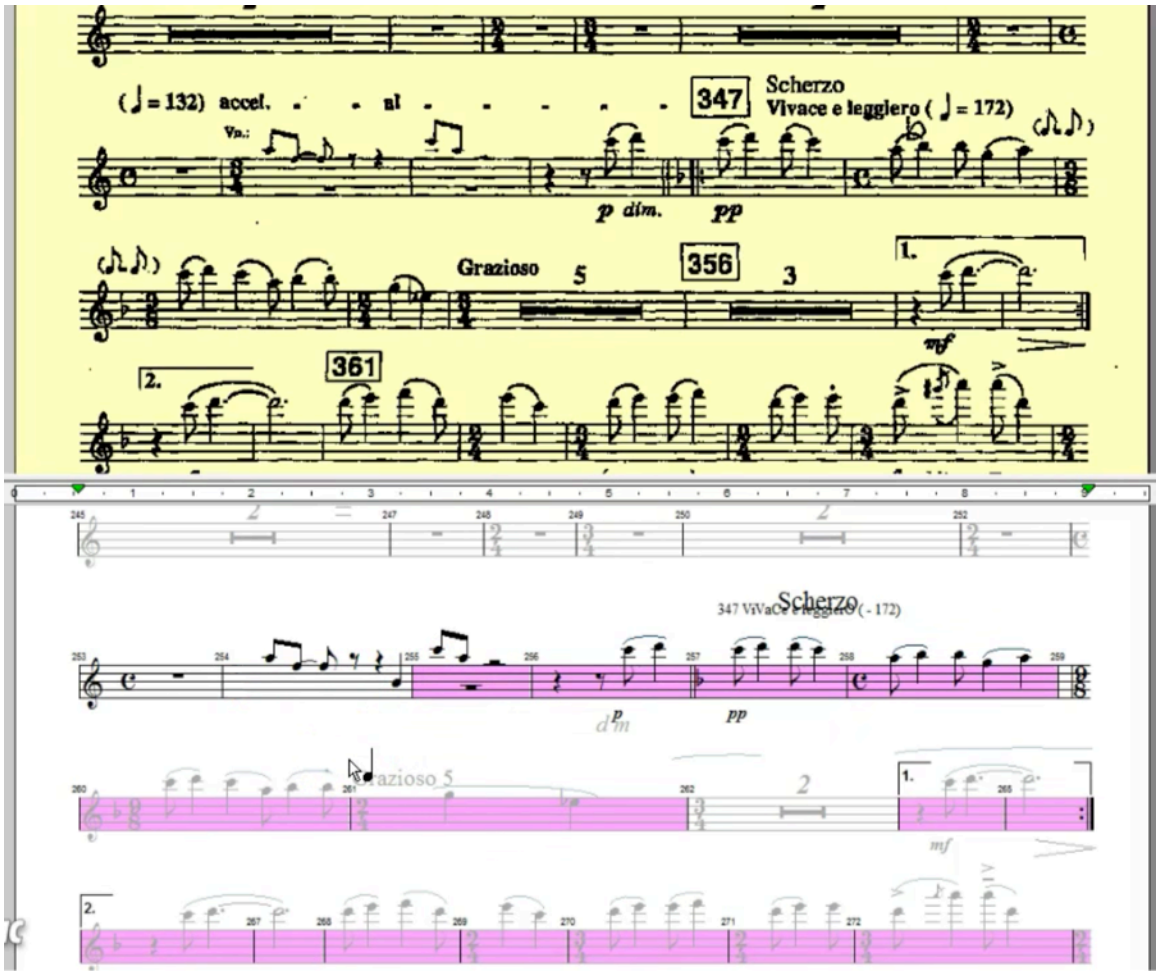
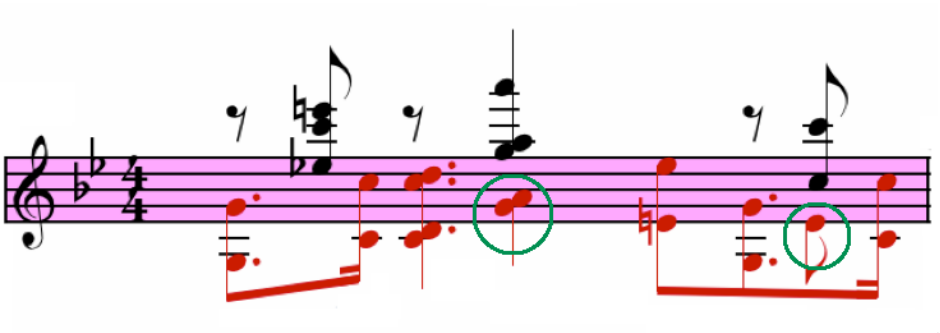


Figure 1.8: The interface for SmartScore. The upper part is the original score image. The lower part is the recognition result shown in a music notation editor. The system highlights the measure in pink to remind users of possible errors.

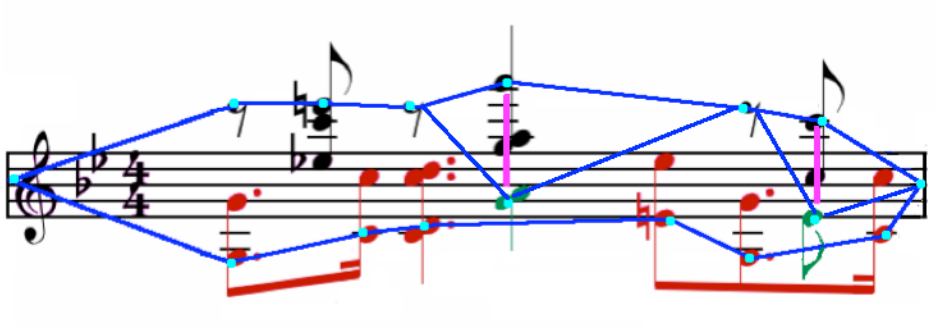
Such errors could come from either the recognition or the interpretation. Recognition errors include missing symbols, misrecognized symbols, extra symbols, etc. Interpretation errors include incorrect pitch or rhythm. Recognition errors usually cause interpretation errors, so they are not totally unrelated. But here we define interpretation errors as cases when the recognition is completely correct but the system still cannot understand the music meaning. We will show two examples of rhythm interpretation error with SmartScore.

1.3.1 Identifying voices in polyphonic music

The first rhythm interpretation error is in a polyphonic measure, as shown in Figure 1.9. SmartScore found two voices in the measure and colored them as red and black. The measure is flagged pink because SmartScore finds that the total duration of the red voice is 11/8, or 3/8 longer than the time signature of 4/4. SmartScore could not make a meaningful interpretation of this example.



(a)



(b)

Figure 1.9: Figure (a) shows the recognition result by SmartScore. SmartScore flags the measure pink because of rhythm contradiction. Figure (b) shows the correct voice labels, voice edges, and coincidence edges.

The problem with this example is that the two circled notes do not belong to either the red voice or the black voice. We color them green to label them as a third

voice, as shown in (b) in Figure 1.9. Now both the black and the red voices have the total duration of 4/4. We can connect the notes with voice edges, and compute their (onset, duration) time.

However, the two green notes do not form a complete voice. Instead, they can be seen as an example of voice splitting and merging of the black voice. To obtain their onset time, we can determine where voices split and merge and then label the voice edges, as shown in (b). Or we can simply apply coincidence edges (pink edges in (b)) according to the vertical alignment rule. Both methods lead to the correct rhythm, while the later one is closer to the way that human musicians read music.

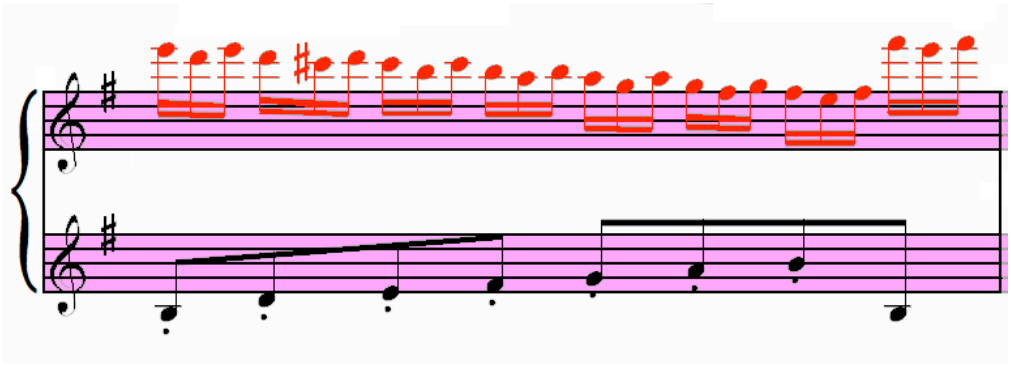
This is a typical rhythm interpretation error resulting from mistakes with identifying multiple voices in a polyphonic measure, especially when there are voices splitting and merging.

1.3.2 Identifying missing triplets

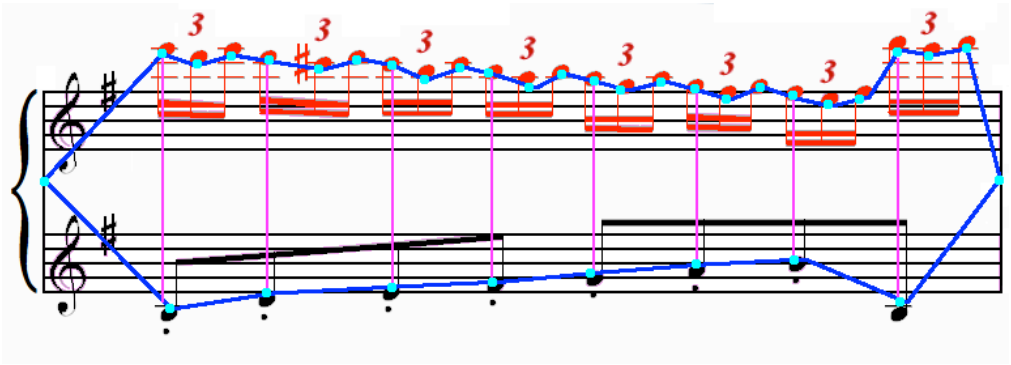
Figure 1.10 shows the second example of rhythm interpretation errors. We mentioned that triplets are usually only labeled in the first measure and omitted in the following measures. In this example, the beam groups on the upper staff are all missing triplets but SmartScore could not automatically recognize that. SmartScore flags the measure because it finds that the duration of the red voice is 3/2, which is longer than the time signature 4/4.

As shown in (b) in Figure 1.10, once we labeled all beamed 16^{th} notes as missing triplets, the total duration of the red voice becomes consistent with the time signature.

It also leads to correct vertical alignment between the black and the red voices (as shown in the pink coincidence edge). Thus we can connect symbols into a connected graph with voice edges and coincidence edges, and compute the consistent onset and duration for both voices.



(a)



(b)

Figure 1.10: Figure (a) shows the recognition result by SmartScore. SmartScore flags the measure pink because the duration of the red voice contradicts the time signature. Figure (b) labels the missing triplet and connects symbols with voice edges and coincidence edges.

This example reveals another rhythm interpretation error; some information such as triplet labels is not written directly in the score but needs to be inferred from constraints of the whole measure. In this thesis, we will focus on solving these rhythm interpretation errors.

1.4 Related work

Optical Music Recognition (OMR) has been studied for decades. In this section, we introduce related literature in OMR in three categories: optical music recognition, optical music interpretation, and evaluation.

1.4.1 Optical music recognition

Most research efforts in OMR are devoted to the recognition step. The first research into OMR began with the thesis of Pruslin [5] and Prerau [6]. Other thesis works devoted to OMR include work by Fujinaga [7] [8], Johansen [9], Rebelo [10] and Bainbridge [11]. The current status of OMR can be found in reviews [1][12][13][14][15] and books [16]–[18].

There are numerous commercial OMR software on the market. The most well-known ones are SharpEye [19], Smartscore [4], Capella-scan [20], PhotoScore [21], etc. Other OMR systems have been built including Ceres [22], Gamera [23], a fuzzy modeling system [24] and Guido [25].

The recognition process can generally be divided into four steps: (1) image pre-processing; (2) staff line detection and removal; (3) primitive symbol segmentation and recognition; (4) compositional symbol assembling. We will briefly introduce each step to give a better understanding of what information we can obtain from the recognition process and how we can use it for interpretation.

The goal of image pre-processing is to make the recognition process more efficient and robust. It may include binarization, noise removal, deskewing, rescaling,

etc. Binarization is a common operation in most OMR systems, and it is a difficult problem, especially for handwritten and degraded music scores. Burgoyne [26] gives a comparative survey of 25 image binarization algorithms on degraded music scores. Pinto [27] presents a binarization method based on music domain knowledge.

The second step is staff line detection and removal. Different approaches have been proposed for detecting and removing staff lines in [28][29][30] [31][32][33][34]. It is a critical step for OMR for four major reasons: (1) The height of staff lines provides the basic scale unit of the score and the scales of other symbols all depends on it. (2) Since staves overlap with most symbols, removing them helps to produce a cleaner image containing only symbols, which helps symbol segmentation and recognition. (3) Because certain symbols (such as rests, clefs, key signatures) have a constant position relation with staff lines, locating staff lines helps to locate these symbols. (4) The staff position that a note head is placed on defines the pitch.

Steps three and four, *primitive* symbol segmentation and recognition and *compositional* symbol assembling, are the core of recognition. The major challenge of music symbol recognition, compared to optical character recognition (OCR), is that it is two-dimensional. While in OCR, characters are treated as a one-dimensional sequence, music symbols vary greatly in shapes and sizes in two-dimensional space.

The most common way of dealing with this problem is through a *bottom-up* approach, which first looks for *primitive* symbols (such as note heads, stems, beams, accidentals, augmentation dots, etc) and then assembles them to *composite* symbols (such as chords or beamed groups), following music notation rules.

Recognition of the primitive symbols can be achieved by different classification algorithms including template matching, K-nearest neighbor, neural network, SVM [35][36][37][38][39][40]. The assemble step usually applies rule-based or heuristic methods based on notation constraints to build the compositional symbols. Other work [41][42] proposed a grammar method to formalize the music notation rules. Rossant [24] constructs the musical notations based on fussy sets and probabilities.

An alternative to the *bottom-up* approach is a *top-down* approach, i.e. using Markov models to represent the transitional state of music knowledge rules [43] [44]. Our OMR system, Ceres [45], also applies a model based top-down approach. The biggest advantage with the top down approach is that it allows simultaneously segmentation, recognition and assembling, which avoids making decisions at early stages of primitive symbol recognition.

The recognition step introduced above provides us with musically meaningful symbols, as well as detailed information including stem directions, beaming, locations on the image, etc. But at this point, we still do not have a high level understanding of rhythms and pitches, which leads to the interpretation step.

1.4.2 Optical music interpretation

Music interpretation seeks to understand the musical meaning of each symbol, including the pitch and rhythm. Very few research efforts have been conducted in this step. However, we found the several research efforts that have applied music notation knowledge at a higher level (in a measure or a page) to improve the result of OMR.

MacMillan [2] developed an OMR system that has a metric consistency step that examines the length of the measure and the vertical alignment. If it finds a contradiction, it tries different corrections to resolve conflicts by considering possible recognition errors, such as extra augmentation dots, confusion between a whole rest and a half rest, etc.

Church [46] proposed a post-processing rhythm correction algorithm using information from similar measures in a piece. It computes the similarity between the current measure and all other measures in the same piece, and uses the most likely one to correct the error.

Several works try to combine the results from multiple OMR systems to improve accuracy. Byrd [47] proposed an approach of "triangulating" the results of several individual systems. Inspired by DNA-sequence alignment, Bugge [48] presented a multiple sequence alignment method on several OMR systems' output to correct pitches and durations errors. Padilla [49] performed alignment of outputs from different OMR programs and performed majority voting to determine the likely recognition.

One common problem with those approaches is that they treat music symbols as one dimensional sequence data, so they mostly work for monophonic music. Instead, we try to solve more complicated rhythm interpretation tasks in polyphonic piano scores. Therefore, we turn to graph structures, which have the virtue of representing the two-dimensional nature of polyphonic music.

In our earlier work [50], we proposed a top-down method that interprets

polyphonic rhythm in terms of a collection of voices. We applied a dynamic programming method that finds the optimal voice label for each symbol in the measure, under the constraints imposed by the time signature. However, some measures cannot be expressed in terms of complete voices.

To solve this problem, we proposed in [51] to represent polyphonic rhythm as a graph. We build the graph by representing symbols with a connected graph, and at the same time check the consistency of the current graph. If we identify any inconsistency in time signature or vertical misalignment, we go back and re-interpret the symbols by considering possible missing tuplet labels. While re-interpretation solves the conflicts; it is difficult to score different re-interpretations and chose the best configuration.

In this thesis, we combine the top-down method and the graph structure in our graph-based algorithm, and introduce this technique in detail in detail in Chapter 4.

1.4.3 Evaluation of optical music recognition

Evaluation for OMR is one of the most important problems. Without an evaluation system, we have no way to know how well an algorithm performs and cannot compare different systems and algorithms. We will discuss two important aspects of evaluation: test dataset and evaluation metric.

The first aspect is dataset. There is no standard dataset for OMR research. Most research builds and tests on a custom made dataset. The main reason for this is that different systems target different scores. For example, some systems only work on single instrument scores, while others work on orchestra scores. Some systems are

made specifically for handwritten scores while others only work on printed scores. Some systems are designed to deal with degraded and old scores with low image quality while other systems only work on cleanly scanned image. Different algorithms also work on music notation with various difficulties from simple monophonic scores to extremely complex piano scores. Byrd [52] proposed a test set in his paper and discussed many important questions.

The second aspect is metric. OMR has many steps including staff removal, primitive symbol detection, compositional symbol assembling and higher level interpretation. Errors can happen in any of these stages. Often previous errors lead to errors in the subsequent step. For example, missing a staff line might cause missing all primitive symbols on that staff line, or missing an augmentation dot might cause a rhythm error for the whole measure.

Different metrics have been proposed. Bainbridge [1] proposed to compute accuracies in four stages: staff line identification, primitive detection, primitive assembly, and music semantics. Droettboom [2] and Bellini [53] suggested two levels of evaluation metrics. A low-level metric reveals the recognition accuracies of primitive symbols (note head, stems, beams, flags and augmentation dots) [54]. A high-level metric evaluates symbols at the note level by their durations and pitches [53]. Szwoch proposed using MusicXML to evaluate accuracy of OMR systems [55].

1.5 Outline of the thesis

The structure of the thesis is as follows. In Chapter 2, we show more examples of

rhythm graphs and discuss the music notation rules that help achieve these graphs. We devote Chapter 3 to the mathematical background of finding the optimized graphs with dynamic programming. The graph-based algorithm is formally introduced in Chapter 4. In Chapter 5 and 6, we propose a monophonic-path method to improve the accuracy and efficiency of the algorithm and introduced an interactive correcting method to correct cases where the automatic method fails. Finally we discuss experiment results that focus on piano scores in Chapter 7 and 8.

2 Music Notation Rules and Violations

In Chapter 1, we introduced the idea of representing rhythm with a rhythm graph. We build the rhythm graph by connecting music symbols with voice edges and coincidence edges. But how can we decide which two symbols are connected by a voice edge or a coincidence edge? Music notation rules help us to do that.

Music notation is a system used to visually represent the aurally perceived music. The goal of music notation is to make the music score easier to understand to readers. Thus notation rules on symbols' spacing, alignment, stem directions and beaming are helpful for rhythm interpretation.

On the other hand, most notation rules can be violated under circumstances. For example, tuplet labels and rests can be omitted to keep scores from overcrowding. Notes can be shifted left or right to prevent overlapping. These are the real challenges we are facing.

In this chapter, we introduce important notation rules that help the system to understand rhythm. We will discuss several examples where notation rules are followed or disobeyed, and how to represent their rhythm with rhythm graphs.

2.1 Time signature rule

The time signature rule says that the total length of a measure is equal to the time signature. The way we enforce this rule through the rhythm graph is that we treat the left and right bar lines as vertices and connect them with other symbols. We label the

(onset, duration) pair of the left bar line as (0,0) and the right bar line as $(T, 0)$, where T is the time signature regarded as a rational number. As shown in the rhythm graph in Figure 2.1, we can see that both voices start from the left bar line and end on the right bar line. According to the onset-duration constraint of voice edges, both voice begin on 0 and end on T, which enforces the time signature rule.

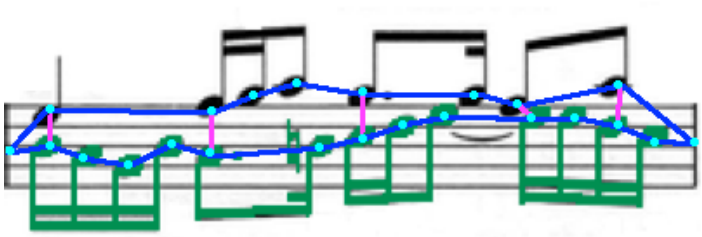


Figure 2.1: A rhythm graph of a polyphonic measure. Time signature is in 4/4.

Figure 2.2 shows an example where the time signature rule seems to be violated. There is only one voice in the measure and the figure shows the only rhythm graph. However, the duration along the voice is $12 \times \frac{1}{8} = \frac{3}{2}$, which contradicts the time signature of the measure, 4/4. The problem is that the beamed 8th notes are missing triplets. Once we identify that they are missing triplets, the time signature rule is still complied with.

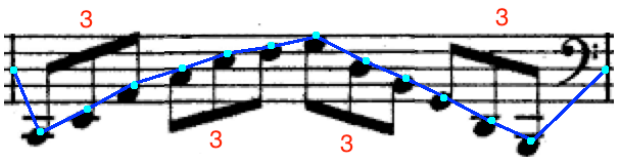


Figure 2.2: A rhythm graph of a measure with missing triplets. Time signature is in 4/4.

However, there are measures where the time signature rule doesn't apply. A *Cadenza* is an improvised passage played by the soloist, usually near the end of a piece. When Cadenza is written out explicitly, it is written in one measure and the rhythm is usually played freely, as shown in Figure 2.3. In such a case, the length of the measure would not be consistent with the time signature and there would not be a unique correct rhythm interpretation of the measure.

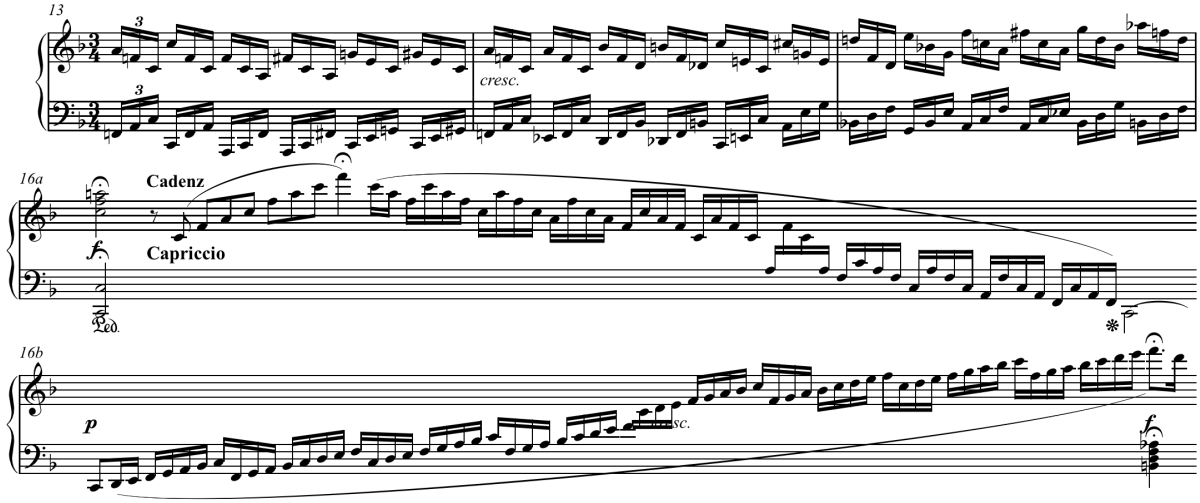


Figure 2.3 An example of a Cadenza.

2.2 Stem direction rule

The stem directions rule provides strong evidence about voices in polyphonic music, where there are at least two voices. The rule states that symbols that belong to the same voice usually have the same stem directions. In terms of a rhythm graph, we can say that a voice edge tends to connect two notes with the same stem direction rather than the opposite directions. Figure 2.1 shows a typical example where all symbols connected by the voice edges have the same stem direction.

Stem direction rule can be violated when two voices merge into one or when one voice splits into two. In Figure 2.4, when two voices with opposite stem directions merge into one voice, one of the voice edges must connect notes with opposite stem directions. Furthermore, when the voice becomes monophonic and stem direction no longer carries voice information, this rule also no longer applies.

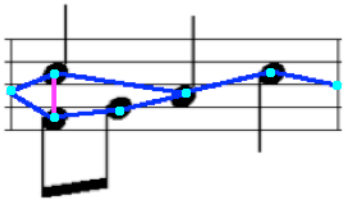


Figure 2.4: A measure shows a violation of the stem direction rule, where a voice edge connects two quarter notes with opposite stem directions.

2.3 Voice-stay-within-staff rule

This rule says that the notes of a voice are more likely to stay within one staff, which means that voice edges are more likely to connect two symbols that belong to the same staff. In piano scores, a system measure, or a grand staff measure, usually has two voices written separately in two staves. The right hand voice is usually written in the upper staff, and the left hand voice is written in lower staff. This makes it easier for the pianist to read and decide which notes each hand plays. Figure 2.5 shows an example of piano score voices separated by staff lines.

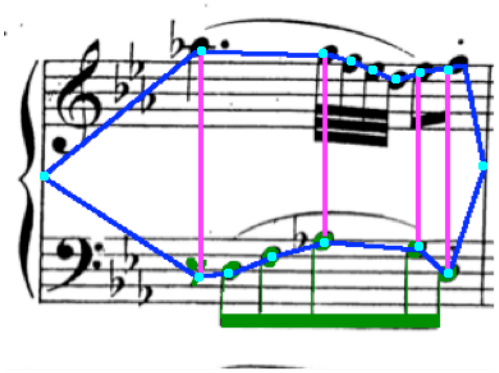


Figure 2.5: An example of a system measure where voices are separated by staff lines.

However, Figure 2.6 (originally from [1]) shows an interesting example where voices are not separated by staff lines, but by stem directions. In this example, apparently the stem directions of the notes indicate that the melody is played using alternating left and right hands. An alternative way to notate this measure is to put the right hand voice in the upper staff and the left hand voice in the lower staff, but that would lead to many ledge lines in both staves and make the score difficult to read. Therefore, we should always consider the possibility of a voice jumping between staves, so we build a rhythm graph on a system measure rather than on a single staff measure.

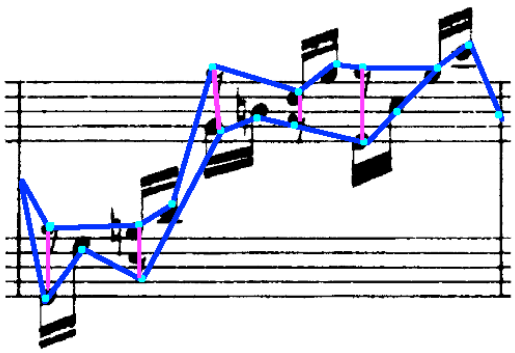


Figure 2.6: An example with a voice jumping between the upper and lower staff, which shows that voice edges can connect two notes belong to two staves, from [1].

2.4 Voice-stays-within-beam rule

Similar to the previous rule, this rule also helps us identify the voice edges. This rule states that a voice usually connects consecutive notes in a beamed group. The aim of beaming is to group notes together, often in complete beats, so that it is easier for musicians to read. Therefore, we are more likely to put a voice edge between two consecutive notes that belong to the same beamed group in a rhythm graph, as shown in Figure 2.5.

However, we have also observed that voices can "jump" in and out of a beamed group. In Figure 2.7, four beamed groups actually belong to one voice. Beaming here no longer indicatess the flow of a voice, but is used to emphasize the proceeding of the scale. Thus, a note does not always follow the preceding note in the same beamed group, this is reason for treating a beamed group with multiple chords as separate vertices in our graph.

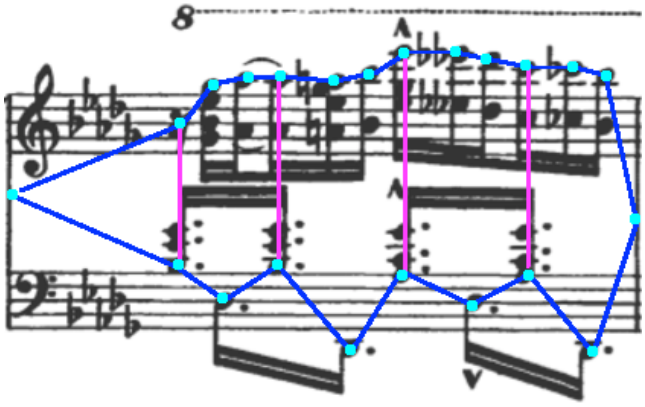


Figure 2.7: An example showing a voice jump between two beams in the lower staff. It shows that a note does not always follow the neighboring note in the same beamed group.

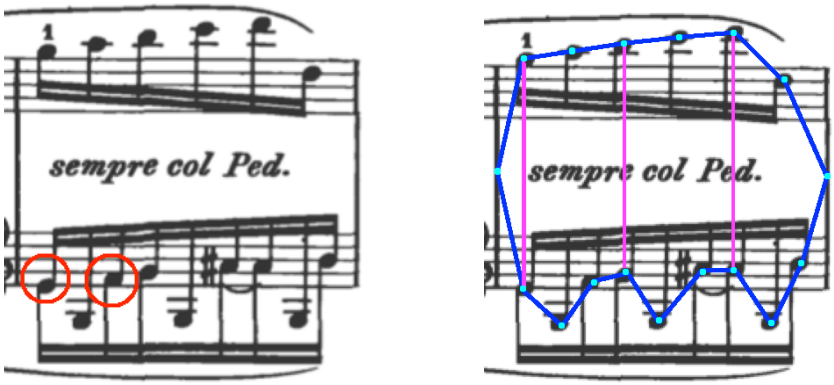
Another example that violates the voice-stays-within-beam rule is *impossible rhythm* [56]. Impossible rhythm happens when two beamed groups share more than

one note head. Sometimes the nominal duration of two beamed notes may lead to onset contradiction on the shared note heads.

Figure 2.8 shows a typical example of impossible rhythm. The stem up voice has triplets and the stem down voice has duplet notes. This leads to a conflicting onset time on the second shared note head (as circled in Figure 2.8 a). The stem up note will have an onset time of 1/12 and the stem down note will have an onset time of 1/16.

There are multiple ways to interpret impossible rhythm. Hook [56] presents a detailed study on examples and choices of interpretations. Here we just simplify the problem by considering two options, *triplet priority* or *duplet priority* interpretation. In this example, triplet priority plays the stem up notes in triplet time and moves the stem down notes to match with the onset of triplet rhythm. In duplet priority, the duplet note is played strictly in tuple time while the first two 16th notes of triplet voice are played as a 32nd notes, so that the third 16th note matches the onset of the duplet note. This interpretation follows the theory that the tuplet voice has a coherent melody.

Here we just plot the rhythm graph of the triplet priority interpretation in (b) in Figure 2.8.



(a) (b)

Figure 2.8: Figure (a) shows an example of impossible rhythm. Figure (b) shows the rhythm graph of the triplet priority interpretation.

Figure 2.9 shows another example of more complicated impossible rhythm. This example may seem daunting even for a human. In the upper staff in each measure, there are nine beamed groups, each sharing note heads with other beamed groups. One of the conflict onsets happens on the circled notes, where the normal stem up 16th note shares the note head with the (missing) triplet stem down note.

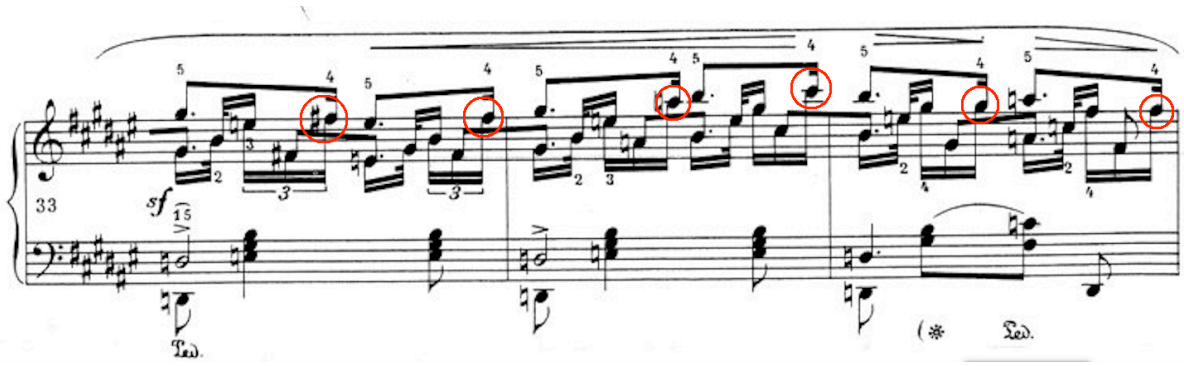


Figure 2.9: Another example of impossible rhythm.

2.5 Ordering rule

The ordering rule states that notes are played in the same order as their horizontal

position in the measure. The vertical alignment rule is a special case of this rule that when two notes align vertically on the score, they usually have the same onset time. In terms of rhythm graphs, coincidence edges are more likely to connect two symbols that align vertically.

It's worth mentioning that when two notes have the same onset time, they do not necessarily align vertically. In many cases, symbols that are supposed to align can be moved to avoid overlapping on the score. As shown in figure 2.10, the four circled notes all have onset time of 0 but are not aligned because notes are shifted to avoid overlapping. In this case, we should also consider coincidence edge between notes that are close but not aligned.

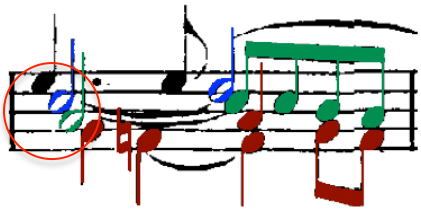
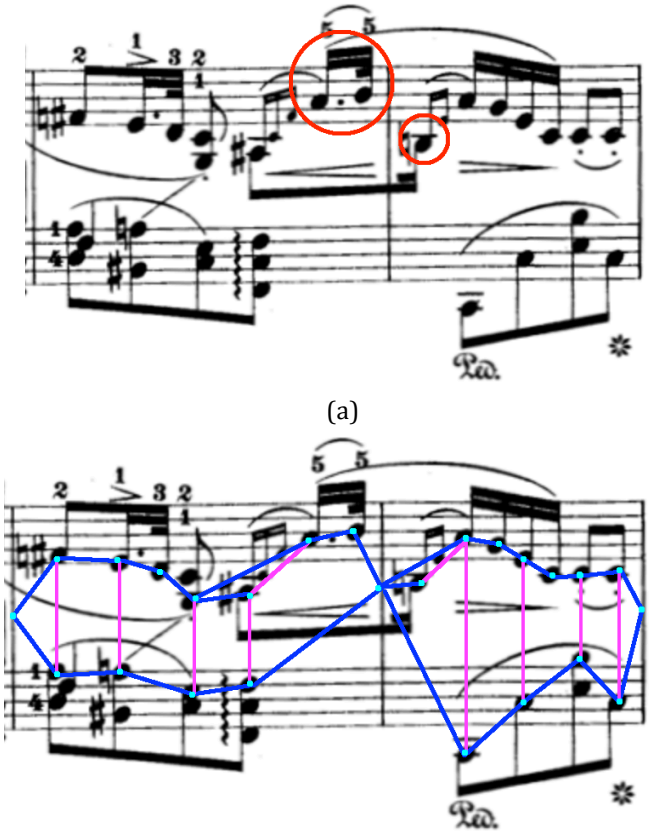


Figure 2.10: A measure consists of four voices, which are colored differently. The first four notes have the same onset time although they do not align vertically.

Figure 2.11 shows another violation of the ordering rule. The example is originally from [57]. The time signature is in 2/4. If we follow the ordering rule, then the circled beamed note in the left measure is played *after* the stem down eighth note on the left, which makes the measure an eighth note too long. The same happens in the right measure: the circled 16th note is played *before* the neighboring beamed note on the right and the measure is a 16th note too long.

Actually this is another example where coincident notes are moved; except that this time, notes are not moved to avoid overlapping, but because of the grace notes between them. A rhythm graph of the measure is shown in (b). We can see that two notes can have coincident onset even they are distant in horizontal position.



(a)

(b)

Figure 2.11: An example where the ordering rule is violated. Time signature is in 2/4. Figure (b) shows the corresponding rhythm graph, from[57].

2.6 Modeling notation rules

From the previous examples, we can see that the biggest challenge in rhythm interpretation is that there is always an exception. The violation examples we show are just a small portion; more extreme examples can be found in the papers by Byrd [57]–[60]. Notation rules can be violated and symbols can be omitted. Somehow music

meaning is still unambiguous to human musicians in most cases, so we hope our system can reach the same interpretation.

In order to deal with these cases, we need a system that not only considers all notation rules at the same time, but also is flexible enough to deal with exceptions. Rule-based system or heuristic method will fail at these cases. Instead, we propose a graph-based model to model these notation rules. More specifically, we define scoring functions on voice edges and coincidence edges based on these notation rules. For example, two notes that align vertically get a higher score when they are connected by a coincidence edge; two notes with the same stem direction have a higher score when connected by a voice edge than by a coincidence edge. And instead of making decision locally based on the score, we choose the graph with the highest score by computing the sum of scores on a whole graph. In this way, we formulate the problem as an optimization problem on a graph. We introduce in chapter 3 the mathematical background of optimization algorithms on graphs.

3 Mathematical Background

In this chapter, we introduce dynamic programming, the optimization technique we use to find the optimal rhythm graph. The organization of this chapter is as follows. We first introduce the background of the dynamic programming technique. In Section 3.2, we present dynamic programming on a sequence-structured graph, and then in Section 3.3 we address the optimization problem on more general graphs.

3.1 Introduction

Dynamic programming [61] is a powerful discrete optimization technique. The basic idea of dynamic programming is to break the original problem into a set of overlapping sub-problems, by remembers the solutions of smaller sub-problems and reusing them to solve the larger problems. Thus each sub problem is solved only once, which keeps the algorithm efficient [62].

Dynamic programming has been widely applied to different areas such as curve detection and stereo matching in computer vision [63], speech recognition [64] and various problems in music informatics including audio segmentation [65], beat tracking [66], score following [67], etc.

These problems are usually modeled with probabilistic graphical models, where dynamic programming is applied to find the most likely assignment of all or some subset of latent variables. The Viterbi algorithm is a famous example that uses dynamic programming for finding the mostly likely states in a sequence [68] and it can be generalized to the max-product algorithm on tree-structured graphs [69]. Dynamic

programming is also used in more general graphs with loops, together with tree decomposition. The essential idea of tree decomposition is to eliminate the loops in a graph and convert it into a junction tree or clique tree. Work by Lauritzen and Spiegelhater [70], Shafer [71][72], Dawid [73], and Andersen [74] applied tree decomposition on probabilistic graphical systems and solved the inference problems with belief propagation, the underlying technique of which is also dynamic programming.

Dynamic programming is also used for optimal decision-making on non-probabilistic graph problems. Work by Bodlaender [75], Bern [76], Arnborg [77] observed that many NP-hard problems (such as maximum matching problem, the traveling salesman problems, maximum independent set and minimum dominating set problems, etc.) can be solved efficiently with dynamic programming in polynomial time for graphs of bounded width (or partial k-trees), with tree decomposition.

In this thesis, we formulate the rhythm interpretation problem as a constrained optimization on a graph. We apply tree decomposition to convert a general graph into a sequence structured clique tree and then solve the optimization efficiently with dynamic programming.

3.2 Dynamic programming on a sequence

We first present a typical application of dynamic programming in solving optimization on a sequence-structured data. Assume we have a sequence of n nodes that we want to label with a state space N, as shown in Figure 3.1. We define $(x_1, \dots, x_n)$ as a certain

configuration that assigns label $x_i \in N$ to the ith element of the sequence. Then we can define an objective function as:

$$H(x_1, \dots, x_n) = \sum_{i=1}^{n} D(x_i) + \sum_{i=2}^{n} V(x_{i-1}, x_i)$$

The score function has two parts. $D(x_i)$ is the score for assigning label x_i to the ith element of the sequence, and $V(x_{i-1}, x_i)$ is the score for assigning two specific labels to a pair of neighboring elements.

Our goal is to find the labeling of the sequence that maximizes the objective function $H(x_1, \dots, x_n)$. We can apply dynamic programming to do that. We begin from the first element and work towards the last element. For each element, we examine the states on the left and compute the optimal score for the current element by reusing the previously solved sub-problem. More specifically, we define

$$\widehat{H}_i(x_i) = \max_{x_1,\dots,x_{i-1}} \left(\sum_{j=1}^{i} D(x_j) + \sum_{j=2}^{i} V(x_{j-1}, x_j)\right)$$

as the score of the best assignment of labels to the first i elements under the constraints that the ith element is labeled as x_i. $\widehat{H}_i(x_i)$ can further be written as a recursive form:

$$\widehat{H}_1(x_1) = D(x_1)$$

$$\widehat{H}_i(x_i) = D(x_i) + \max_{x_{i-1}} (\widehat{H}_{i-1}(x_{i-1}) + V(x_{i-1}, x_i))$$

We gradually fill the table with $\widehat{H}_i(x_i)$ from left to right, as in Figure 3.1. Once all of the $\widehat{H}_i(x_i)$ are computed, the optimal solution of the original objective function

$H(x_1, \dots, x_n)$ is found by taking the maximum of the last state $\widehat{H}_n(x_n)$. In order to get the optimal label of each element $x_1^*, x_2^*, \dots, x_n^*$, we can just trace back according the following equations:

$$x_n^* = \arg\max_{x_n} \widehat{H}_n(x_n)$$

$$x_{i-1}^* = \arg\max_{x_{i-1}} \widehat{H}_{i-1}(x_{i-1}) + V(x_{i-1}, x_i^*) \qquad \text{for } i = n, n-1, \dots$$

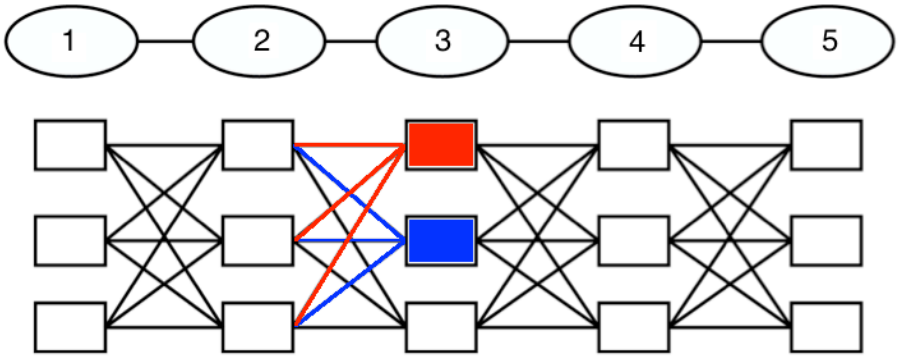


Figure 3.1: An example of applying dynamic programming on sequence-based data. Each state is determined by the solutions of previous states. The process of computation with dynamic programming can be seen as filling out the table from left to right.

3.3 Generalization

The optimization method with dynamic programming is not limited to sequence-structured data, but can be generalized to graphs. Given a graph $G = (V, E)$, where V represents the vertices and E represents the edges, we associate G with random variable x. We also define an objective function $H(x)$ on the graph,

$$H(x) = \sum_{v \in V} H_v(x_v) + \sum_{e \in E} H_e(x_e)$$

where $H_v(x_v)$ is the score for assigning label x_v to a vertex v, and $H_e(x_e)$ is the score for assigning certain labels to an edge e. Our goal is to find the best configuration that maximizes $H(x)$.

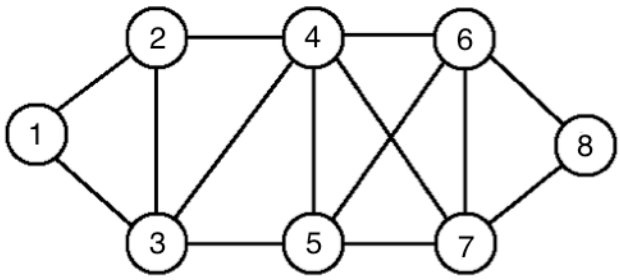


Figure 3.2: A general graph consisting of G = (V, E).

Figure 3.2 shows an example of one such graph. We label all vertices with $v_1, \dots, v_8$, as shown in the figure and try to find max $H(x_1, \dots, x_8)$. We observe that the only vertices connected to v_8 are v_6 and v_7. In other words, set $\{x_{v_6}, x_{v_7}\}$ creates a *separator* that separates $\{x_{v_8}\}$ from all vertices in $\{x_{v_1}, \dots, x_{v_5}\}$. Variable x_{v_8} only depends on the values of $\{x_{v_6}, x_{v_7}\}$ and the labels of edges, $e_{6,8}$ and $e_{7,8}$.

We try to solve the optimization problem in the following way. We define $x_{1:k}$ as a configuration of variables on vertices and edges that concerns only the first k vertices, and $H(x_{1:k})$ as the score on corresponding vertices and edges. More specifically,

$$x_{1:k} = \{x_{v_1}, x_{v_2}, \dots, x_{v_k}\} \cup \{x_e : e = (v_i, v_j), i < j \leq k\}.$$

$$H(x_{1:k}) = \sum_{v \in \{v_1, v_2, \dots, v_k\}} H_v(x_v) + \sum_{e \in \{e=(v_i, v_j), i<j\leq k\}} H_e(x_e) .$$

We can also define $\widehat{H}(x_{v_8}) = \max_{\{x_{1:8}\}/\{x_{v_8}\}} H(x_{1:8})$ as the maximal value of $H(x_{1:8})$

under the constraint that v_8 is labeled as x_{v_8}. Similarly, by defining $\widehat{H}(x_{v_6}, x_{v_7}) = \max_{\{x_{1:7}\}/\{x_{v_6}, x_{v_7}\}} H(x_{1:7})$, we can write the following recursive function:

$$\widehat{H}(x_{v_8}) = \max_{\{x_{v_6}, x_{v_7}, x_{e_{6,8}}, x_{e_{7,8}}\}} \widehat{H}(x_{v_6}, x_{v_7}) + H_v(x_{v_8}) + H_e(x_{e_{6,8}}) + H_e(x_{e_{7,8}})$$

Based on this recursion form, we break the larger problem into a smaller sub-problem, which is the basic idea of dynamic programming. In the following section, we will discuss in detail how to use *graph decomposition* and dynamic programming to solve optimization on graphs.

3.3.1 Graph decomposition

The goal of graph decomposition is to eliminate loops by grouping vertices into *cliques*. Given a graph $G = (V, E)$, we can associate the graph with an objective function $H(x_1, \dots, x_n)$. To decompose a graph, we denote a list of subsets $C = \{C_1, C_2, \dots, C_n \mid C_i \subset V\}$. We refer to these subsets as *cliques*, which means that they are fully connected sets of vertices. The union of these cliques is V, and they are usually not disjoint. We define the value of variables C as x_C. We also rewrite the objective function of $H(x_1, \dots, x_n)$ into a sum of the function on cliques, $H_n(x_{C_n})$, such that:

$$H(x_1, \dots, x_n) = \sum_{i=1}^{N} H_n(x_{C_n})$$

For simplicity of explanation, the score function $H(x_1, \dots, x_n)$ and $H_n(x_{C_n})$ here only concerns score on the vertices, but it can be easily generalized to scores for both vertices and edges.

We also denote a list of *separators*, $\{S_1, S_2, \dots, S_{n-1}\}$. A separator S_i satisfies $S_i = C_i \cap C_j$. The sequence of overlapping cliques satisfies that for any k > 1, the intersection of C_k with the union of all previous cliques C_j is contained in S_{k-1}. That is, S_{k-1} separates clique C_k from all previous cliques C_j , and there is no longer any edge that connect C_k to previous cliques.

An example of a clique sequence of the graph in Figure 3.2 is shown in Figure 3.3. In this way, we decompose a graph into a clique sequence. However, it is worth mentioning that the graph decomposition is by no means unique; Figure 3.3 only shows one example.

Figure 3.3: The clique sequence converted from the graph in Figure 3.2.

3.3.2 Dynamic programming on a clique graph

Now that we have achieved a clique graph, we apply dynamic programming to obtain the best configuration in a similar way as we did for a sequence-structured graph in Figure 3.1. To put it briefly, since a separator S_{k-1} separates C_k from all previous cliques, the value of separator S_k only depends on S_{k-1} and C_k. We use this relation to recursively solve a larger problem with the solution to smaller sub-problems.

More specifically, we define variable on S as x_s and $\widehat{H}_i(x_{S_i})$ as the score of the best assignment of labels to all the elements in the first i cliques in the sequence with

the constraints that the elements in the ith separator is labeled as x_{S_i} . Then we have a recursive function,

$$\widehat{H}_1(x_{S_1}) = \max_{x_{C_1 \setminus S_1}} H_1(x_{C_1})$$

$$\widehat{H}_i(x_{S_i}) = \max_{x_{C_i \setminus S_i}} \widehat{H}_{i-1}(x_{S_{i-1}}) + H_i(x_{C_i})$$

Once all $\widehat{H}_i(x_{S_i})$ are computed, the maximum value of H is:

$$\max H(x_1, \dots, x_n) = \max_{x_{C_N}} \widehat{H}_{N-1}(x_{S_{N-1}}) + H_N(x_{C_N})$$

And the labeling of each element in cliques can be traced back with:

$$x^*_{C_N} = \arg\max_{x_{C_N}} \widehat{H}_{N-1}(x_{S_{N-1}}) + H_N(x_{C_N})$$

$$x^*_{C_i \setminus S_i} = \arg\max_{x_{C_i \setminus S_i}} \widehat{H}_{i-1}(x_{S_{i-1}}) + H_i(x_{C_i \setminus S_i}, x^*_{S_i})$$

$$x^*_{C_1 \setminus S_1} = \arg\max_{x_{C_1 \setminus S_1}} \widehat{H}_i(x_{C_1 \setminus S_1}, x^*_{S_1})$$

with the middle equation holding for $i = N-1, \dots, 2$

In the next chapter, we will introduce the formal definition of a rhythm graph and the details of how we apply dynamic programming to interpret rhythm.

4 Graph-based Rhythm Interpretation

In the previous two chapters, we discussed the music notation rules and the optimization method for rhythm interpretation. In this chapter, we formally introduce our graph-based rhythm interpretation algorithm. We represent rhythm as a connected graph on all the symbols in a measure. We call this graph the *rhythm graph*. Our approach represents the notes and rests in a system measure as the vertices of a graph. We connect the graph by adding voice edges and coincidence edges between pairs of vertices. The graph is constructed under the constraint that it leads to a meaningful rhythm interpretation, which will be discussed in detail in this section. We score the graph based on music notation rules and choose the graph that has the best score. The problem is thus converted into a constrained optimization problem of finding the graph with the highest score. The rhythmic interpretation follows simply from the connected rhythm graph.

4.1 Vertices

The input to our system is a list of rhythm-bearing symbols in a system measure, containing their types, locations in the measure, and stem-directions. These symbols are obtained from the recognition step of our OMR system, Ceres [41]. The system provides human-computer interactions that enable user-supplied information to make sure the symbols are complete and correct [78].

We treat the list of input symbols as vertices in our rhythm graph, which we denote as V. The vertex set V include all notes and rests as well as the bar lines in a

system measure. Here a note is either a single note or a chord. A beamed group consisting of four chords is considered as four vertices. We order the vertices based on their horizontal positions in the measure from left to right, breaking ties arbitrarily, and label them as $v_1, v_2, \ldots, v_M$. This is the order in which we visit vertices in our rhythm graph algorithm, which will be discussed later.

Each vertex $v \in V$ has a *nominal* duration $d(v) \in \mathbb{Q}^+$ (the non-negative rationals). For example, a dotted eighth note has duration of $3/16$. Bar lines have a duration of zero. The actual durations of vertices are usually the same as the nominal duration, but can be different when they are tuplets. For example, an eighth note will have duration of $1/12$ when it is a triplet.

4.2 Edges

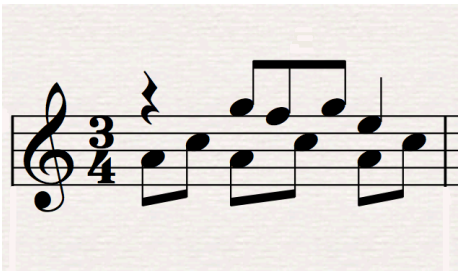
As for edges, we define two types of edges, voice edges and coincidence edges. A voice edge connects two vertices that are played consecutively. A coincidence edge connects vertices that are played at the same time. For a given measure, we choose a set of *possible* voice edges, E_v, which is a subset of all possible edges in $V \times V$. We will make binary determinations in each edge in the possible edge set, E_v. We define a similar possible coincidence edge set, E_c. And we define the whole possible edge set $E = E_v \cup E_c$.

The choice of E_v and E_c is important for our algorithm. Edges that are not in the possible edge set will never be considered as possible edges. Ideally E_v and E_c would include all edges in the graph, as long as allowed by music notation rules. But for

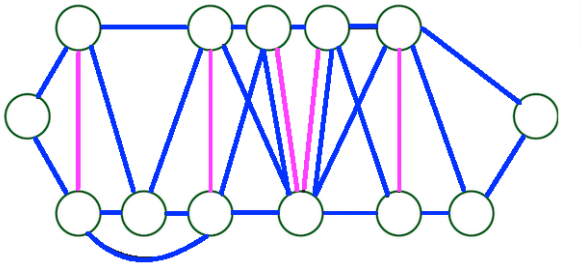
computation reasons, we only consider the edges that are more likely to be in the rhythm graph. Here, we just introduce a heuristic method of choosing the possible edge set. For instance, edges that connect vertices that are closer but not vertically aligned are more likely to be voice edges. Edges that connect vertices that are vertically aligned are more likely to be coincidence edges. The heuristic method to pick possible edges is somewhat naïve and has its drawbacks. We will improve this method in Chapter 5.

Recall that all vertices are numbered from left to right as $v_1, v_2, v_3, \dots, v_n$, and we assume that each edge $e = (v_i, v_j) \in E$ has $i < j$. For a voice edge $e \in E_v$, $e = (v_i, v_j)$ means that vertex v_j follows v_i in the voice. Since we have ordered vertices from left to right, this voice direction matches with their horizontal order in the measure. In our algorithm, we assume that a note with lower index cannot be played after a note with higher index. This rule might be violated rarely in some cases, but we just assume it is correct in our model. For a coincidence edge, $e \in E_c$, since the edge's meaning does not depend on the direction of the edge, so the two vertices can be written in either order with the same meaning. So $i < j$ in $e = (v_i, v_j)$ places no constraint.

Figure 4.1 gives an example of a measure and it is corresponding possible edge set. Notes, rests and barlines are represented as vertices (circle). Possible voice edges, E_v, are colored as blue. Possible coincidence edge, E_c, are colored as magenta. Notice that sets E_v and E_c can overlap.



(a)



(b)

Figure 4.1: Figure (a) is a polyphonic measure and (b) shows its corresponding vertices and possible edge set. Each circle in (b) represents a note, a rest or a barline. Blue edges represent the possible voice edges. Magenta edges represent the possible coincidence edges.

4.3 Notation of variables

Given the vertices and the possible edge set, we can begin building the rhythm graph. Before introducing the algorithm, we first define some notation for the variables we are going to use.

To interpret the rhythm, we need to label the onset and duration of each vertex $v \in V$. We define a vertex variable $x_v = (x_v^o, x_v^t)$, where $x_v^o \in \mathbb{Q}^+$ gives the onset time and $x_v^t \in \{0,1\}$ labels the triplet status, which is 1 if and only if v is a missing triplet. For now we only consider the case of missing triplet for clarity of explanation. However, it can be extended to other tuple values by changing the binary labels to

multiple labels.

Also we want to choose a subset of edges from possible edge set E to specify our rhythm graph. For each edge $e \in E$, we define an edge variable $x_e \in \{0,1\}$, describing whether this edge exists in the rhythm graph or not.

We define a vector $x = \{x_e\} \cup \{x_v\}$, $e \in E$ and $v \in V$, for the whole configuration of a rhythm graph. A configuration of x defines a certain rhythm interpretation of the measure. Figure 4.2 shows an example of the desired rhythm graph for the measure in Figure 4.1. Only edges that are labeled as $x_e = 1$ are drawn in the graph. White circles are vertices labeled as non-triplet, while green circles are vertices labeled as triplet notes.

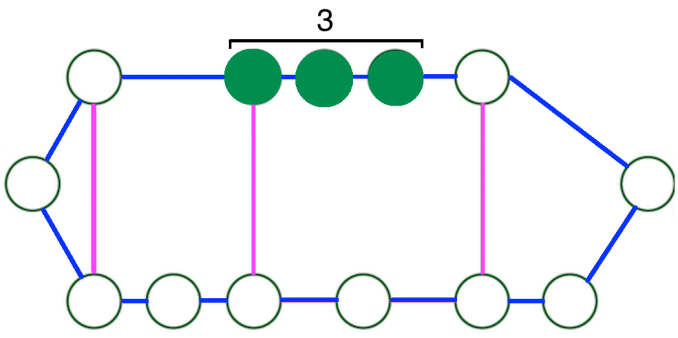


Figure 4.2: The desired rhythm graph of the measure in Figure 4.1. Voice edges are labeled as blue. Coincidence edges are labeled as magenta. Solid circle vertices are labeled as triplet notes. Open circle vertices are labeled as non-triplet notes.

4.4 Legal graphs

Not all configurations of x lead to meaningful rhythmic interpretations. We define a configuration of x as *legal*, denoted as $x \in L$, if it satisfies the following constraints:

1. $x_v^o = 0$ when v is the left barline, and $x_v^o = T$ when v is the right barline, where T is the time signature $T \in \mathbb{Q}^+$. This constraint is to satisfy the time signature constraints rule.

2. If $x_e = 1$ for $e = (v_i, v_j) \in E_v$, then $x_{v_j}^o = x_{v_i}^o + D(x_{v_i}^t)$, where

$$D(x_{v_i}^t) = \begin{cases} d(v_i), & if\ x_v^t = 0 \\ \frac{2}{3} d(v_i), & if\ x_v^t = 1 \end{cases}$$

where $d(v_i)$ is the nominal duration of vertex v_i.

3. If $x_e = 1$ for $e = (v_i, v_j) \in E_c$, then $x_{v_j}^o = x_{v_i}^o$.

4. For each $v_j \in V$ other than the left barline, there is at least one edge $e = (v_i, v_j)$ with $x_e = 1$. This ensures that the onset of v_j is set. It also ensures that v_j is connected to the graph.

5. For each $v_j \in V$, if there are multiple edges $e = (v_i, v_j)$ with $x_e = 1$, they must lead to a unique onset time $x_{v_j}^o$. Onset x_v^o follows constraint (2) and (3), depending on the type of the edges.

These legal graph constraints prohibit many illegal configurations of x that do not lead to meaningful rhythm. However, we will still be left with a great number of legal configurations that correspond to different rhythms. To find the best legal configuration, in the next section we define an objective function to score the graphs based on music notation rules.

4.5 Objective function

We define an objective function to score a rhythm graph on the variable of $x = \{x_e\} \cup \{x_v\}$,

$$G(x) = \sum_{v \in V} G_v(x_v) + \sum_{e \in E} G_e(x_e)$$

where $x_v = (x_v^o, x_v^t)$ defines the rational onset and triplet labeling of a vertex $v \in V$, $x_e \in \{0,1\}$ defines whether an edge $e \in E$ exists or not, $G_v(x_v)$ scores a vertex variable , and $G_e(x_e)$ describes the plausibility of a particular edge.

The vertex score, $G_v(x_v)$, is a negative score. $G_v(x_v)$ can be divided into two parts. The first part penalizes labeling a vertex as triplet ($x_v^t = 1$), because we prefer simple rhythm interpretation. The second part penalizes complex onset time for x_v^o. This is also because we prefer simple onset time such as $1/4$, $1/2$, $3/8$, $7/16$. Because an onset time of $23/36$ or $37/54$ usually implies that we might falsely label a non-triplet symbol as a triplet. Thus, such complex onsets receive a larger penalty on score.

The edge score, $G_e(x_e)$, is determined by the edge type of $e = (v_i, v_j)$ and the properties of vertices v_i and v_j. We define a function $G_e(x_e)$ on music notation "rules" for interpreting rhythm. For example, if v_i and v_j have the same stem directions and are close in horizontal position (but not aligned), then $G_e(x_e), e = (v_i, v_j)$ for $e \in E_v$ will have a higher score. If v_i and v_j have different stem directions and are almost aligned, then $G_e(x_e), e = (v_i, v_j)$ for $e \in E_c$will have a higher score. Score $G_e(x_e)$ can be either positive or negative.

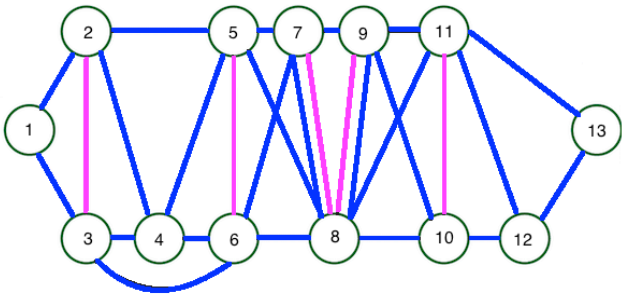
The score of a certain edge or a vertex tells how plausible a particular label is. However, we cannot label each edge or vertex independently according to its own optimal score since that could lead to illegal rhythm graphs. Thus we need to solve the constrained optimization problem such that it produces a legal graph and maximizes the objective function,

$$\hat{x} = \underset{x \in L}{\operatorname{argmax}} G(x)$$

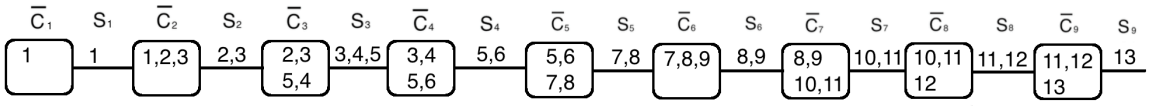
where L is the set of all legal configurations of x.

4.6 Optimization

The brute force method to find the labeling of x that maximizes $G(x)$ is to visit all combinations of vertices and possible edges labels and score all the legal configurations of x. Since each triplet variable x_v^t and edge variable x_e can be labeled as either 0 or 1, the size of configuration of x is 2^{m+n}, where n is the number of vertices and m is the number of possible edges. As $m + n$ can easily reach 60~100, this method is computationally infeasible. Instead we solve the problem with dynamic programming.



(a)



(b)

Figure 4.3: Figure (a) shows the vertices and all possible sets for the measure in figure 4.1. Figure (b) shows the sequence of cliques and separators created with the algorithm.

In order to solve the optimization problem, we want to convert the possible edge set in (a) into the clique sequence as shown in (b). The decomposition algorithm is defined as follows:

Given a set of vertices $V = \{v_1, \dots, v_M\}$, we define $\bar{C}_1 = \{v_1\}$ and $S_1 = \{v_1\}$. We create a list of sets $\{\bar{C}_1, \bar{C}_2, \dots, \bar{C}_N\}$ and $\{S_1, S_2, \dots, S_N\}$ with the following procedure.

1. Begin with set S_{n-1} and let $m = max\{i: v_i \in S_{n-1}\} + 1$. We let $\bar{C}_n = S_{n-1} \cup \{v_m, v_{m+1} \dots, v_k\}$, such that k is the smallest index satisfying $\exists v_i \in \bar{C}_n$, $e = (v_i, v_j) \in E \Rightarrow v_j \in \bar{C}_n$. That is, we keep adding vertices until there is at least a vertex in $\bar{C}_n$ that only connects to vertices in $\bar{C}_n$.

2. Let $R_n = \{v_i \in \bar{C}_n : e = (v_i, v_j) \in E \Rightarrow v_j \in \bar{C}_n\}$. That is, let R_n be all vertices v_i in $\bar{C}_n$that only connect to the vertices v_j in $\bar{C}_n$. We create S_n by removing R_n from $\bar{C}_n$and get $S_n = \bar{C}_n / R_n$.

3. Go back to step 1 and create set $\bar{C}_{n+1}$ with S_n, until $v_M \in \bar{C}_{n+1}$.

4. Set $S_N = \{v_M\}$.

Our algorithm creates a sequence of overlapping sets $\bar{C}_n$ and S_n, as shown in Figure 4.3. We call $\bar{C}_n$ *cliques* and we call S_n *separators,* satisfying $S_n = \bar{C}_n \cap \bar{C}_{n+1}$. Now that we have a sequence structure, it is easy to apply dynamic programming. To do this, we must rewrite our objective function in terms of the sum of potential functions that depend only on variables in the cliques.

Notice that so far our cliques, $\bar{C}_n$, only contain vertex variables, x_v, but our objective function also contains edge variables x_e. Thus we define $T_n = \{e = (v_i, v_j) \in E: v_i \in \bar{S}_{n-1}, v_j \in \bar{C}_n / \bar{S}_{n-1}\}$, which are all edges that connect between vertices in the previous separator $\bar{S}_{n-1}$ and new vertices in incoming clique $\bar{C}_n$. We let $C_n = \bar{C}_n \cup T_n, n = 1,2,\dots,N$. So we keep the structure of the sequence of cliques and separators in Figure 4.3, but change every $\bar{C}_n$ to C_n. Now C_n contains both vertices and edges, while S_n still only contains vertex variables.

We define score functions on the cliques as,

$$H_1(x_{C_1}) = G_v(x_{v_1})$$

$$H_n(x_{C_n}) = \sum_{v_j \in C_n / S_{n-1}} G_v\left(x_{v_j}\right) + \sum_{e_{i,j} \in E_n} G_e\left(x_{e_{i,j}}\right)$$

for $n = 2, \dots, N$.

This definition ensures that each vertex variable, x_v, and edge variable, x_e, appears in exactly one clique function H_n. Thus it allows us write the objective function $G(x)$ in terms of the sum of functions on cliques, $H_n(x_{C_n})$, such that

$$G(x) = \sum_{v \in V} G_v(x_v) + \sum_{e \in E} G_e(x_e) \quad (1)$$

$$= \sum_{n=1}^{N} H_n(x_{C_n})$$

To solve the optimization problem, we define

$$\widehat{H}_n(x_{S_n}) = \max_{\substack{\{x_{1:S_n}\}/x_{S_n} \\ \{x_{1:S_n}\} \in L}} G(x_{1:S_n}),$$

where $x_{1:S_n} = \{x_{v_1}, x_{v_2}, \dots, x_{v_k}\} \cup \{x_e : e = (v_i, v_j), j \leq k\}$, $k = \max\{i : v_i \in S_n\}$. L is the set of all legal configurations of a rhythm graph. Here we extend the legal definition of a graph into cliques, which will be explained in detail shortly.

We can further write $\widehat{H}_n(x_{S_n})$ in the following recursive form,

$$\widehat{H}_1(x_{S_1}) = \max_{x_{C_1/S_1} : x_{C_1} \in L} H_1(x_{C_1})$$

$$\widehat{H}_n(x_{S_n}) = \max_{x_{C_n/S_n} : x_{C_n} \in L} \widehat{H}_{n-1}(x_{S_{n-1}}) + H_n(x_{C_n})$$

where $n = 2, \dots, N$.

Once the $\{\widehat{H}_n(x_{S_n})\}$ are computed, we have

$$\max G(x) = \max_{x_{S_N} \in L} \widehat{H}_N(x_{S_N})$$

And the labeling of x can be traced back with

$$x^*_{S_N} = \operatorname*{argmax}_{x_{S_N} \in L} \widehat{H}_N(x_{S_N})$$

$$x^*_{C_n/S_n} = \operatorname*{argmax}_{x_{C_n/S_n} : x_{C_n} \in L} \widehat{H}_{n-1}\left(x_{S_{n-1}}\right) + H_n(x_{C_n/S_n}, x^*_{S_n})$$

$$x^*_{C_1/S_1} = \operatorname*{argmax}_{x_{C_1/S_1} : x_{C_1} \in L} H_1\left(x_{C_1/S_1}, x^*_{S_1}\right)$$

with the middle equation holding for $n = N, .., 2,$

During the process of dynamic programming, we enforce the legality of the rhythm graph in terms of the legality of clique C_n. Here we explain how.

1. Constraint (1), the time signature constraint, is enforced by adding restrictions on x_{C_1} and x_{C_N}. If x_{C_1} and x_{C_N} satisfy the time signature constraint, the rhythm graph also satisfies the constraint.

2. The construction of the vertex sets C_n ensures that for any vertex v_j, all edges $e = (v_i, v_j)$ are contained in the same clique C_n that v_j belongs to. So constraints (2), (3) and (5) can be enforced as constraints on x_{C_n} such that all edges that connect v_j lead to a unique onset time v_j^o.

3. Constraint (4), the connectedness constraint, is enforced such that for each vertex $v_j \in \{C_n/S_{n-1}\}$, there is at least one edge $e = (v_i, v_j)$ that is labeled as $x_e = 1$. This also ensures that vertex v_j is connected.

By making sure each clique configuration is legal, we make sure the whole configuration of the graph is legal as well.

Finally, we are able to find the best legal configuration of vertices and edges that maximize the objective function $G(x)$ as shown in Figure 4.2, and tracing back on labels of $x_v = (x_v^o, x_v^t)$ gives us the rhythm interpretation.

5 Monophonic path method

In chapter 4, we discussed that the first step of the graph-based rhythm algorithm is to choose a *possible* edge set. The possible edge set has a great impact on how well the later graph-based algorithm works. We will only be picking edges for the rhythm graph from the possible edge set, therefore we might fail to build a rhythm graph if we miss a true edge in this step. The method we introduced previously to generate the possible edge set is a heuristic method based on the notation rules. The biggest drawback of a heuristic method is that it fails in cases when the notation rule is violated, which will be discussed in detail in the following section.

In this chapter, we propose a monophonic path method to choose the possible voice edge set. If we sort the rhythm bearing symbols in a measure from left to right and treat them as a sequence of vertices, $V = \{v_1, \dots, v_n\}$, a monophonic path is a subsequence of V that is connected by voice edges and whose durations add up to the time signature. Finding monophonic paths is computationally cheap, and the edges in the monophonic paths are more likely to "exist" in the rhythm graph. Thus it is more likely to improve our graph-based algorithm.

5.1 Local heuristic method

A heuristic method to pick possible voice edge set is to make local decisions based on different music notation rules. We can score the voice edges between all pairs of vertices and then pick the ones with the highest scores. However, this method has its limitations, which will be shown in the following two examples.

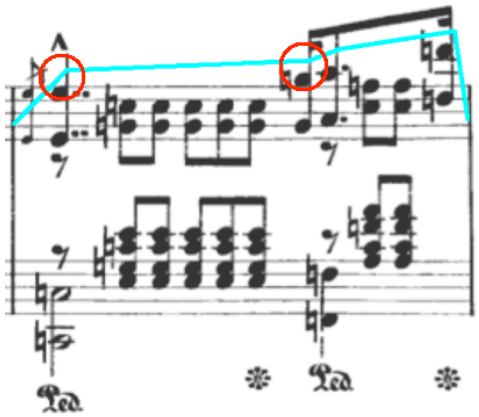


Figure 5.1: An example of a monophonic path showing that two symbols far away can also be connected by a voice edge.

We usually believe that two notes that are horizontally closer but not aligned are more likely to be connected by a voice edge. In other words, the larger the distance between two symbols, the less likely that we connect them with a voice edge. This is not always true. Figure 5.1 shows a monophonic path in the measure. We can see that the two circled notes are connected by a voice edge although they are far away horizontally. If we use the heuristic method, this edge is likely to be missed in the possible voice edge set.

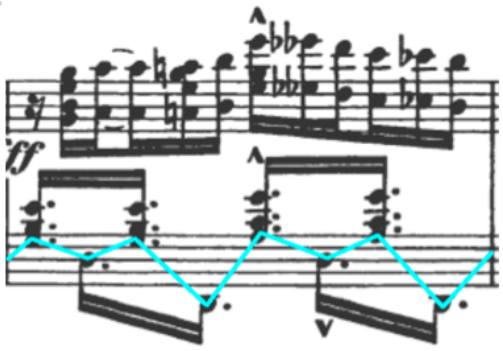


Figure 5.2: An example of a monophonic path showing that voice edges do not always connect notes that are beamed together.

We also tend to believe that consecutive beamed notes are more likely to be connected by voice edges. While this is true for most cases, it can also be violated. Figure 5.2 shows an example where the monophonic voice jumps between two beamed groups with different stem directions.

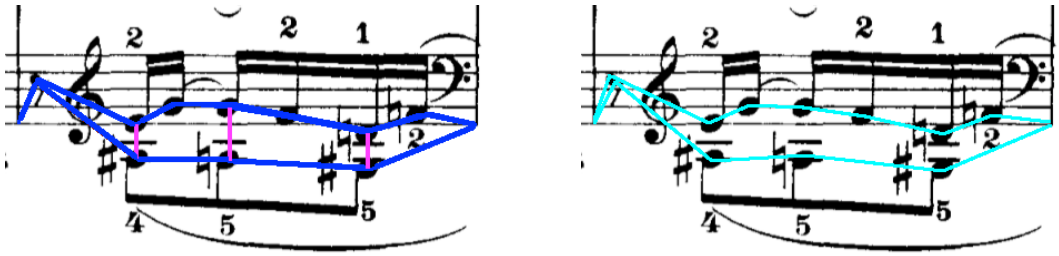
The previous two examples show how a heuristic method fails. If the possible edge set misses the *true* edges that actually exist, then the later algorithm must fail to build a successful rhythm graph. On the other hand, if we include many more edges in the possible edge set to avoid missing *true* edges, it could lead to a very dense graph. A denser graph will lead to larger cliques when we convert the graph to a clique tree, and computational complexity will grow exponentially with the size of the clique.

Therefore, picking a small and accurate possible edge set is crucial to our algorithm. In the following section, we propose a monophonic path method to choose the possible edge set.

5.2 Monophonic path method

A monophonic path is defined as a subsequence of symbols in a measure that is connected by voice edges and whose durations add up to the time signature. In Figure 5.3, the blue edges in (a) shows the true voice edges in the rhythm graph, and (b) shows two monophonic paths with light blue edges. We can see that the voice edges in the polyphonic graph happen to be the same as the voice edges in two monophonic paths. In other words, if a voice edge does not belong to any monophonic voice, it probably

will not appear in the rhythm graph. Therefore, finding the correct monophonic paths gives us a more reliable possible voice edge set for the rhythm graph.



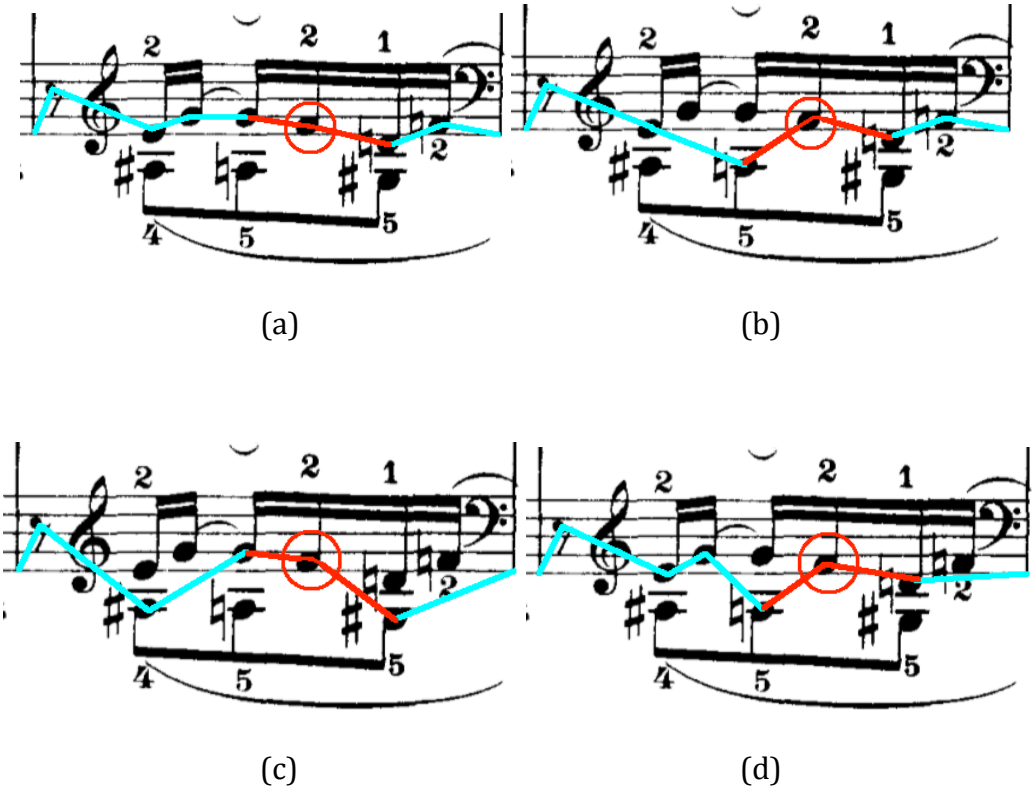
(a) (b)

Figure 5.3: Figure (a) shows a rhythm graph. Figure (b) shows two monophonic voices connected by voice edges.

Finding monophonic paths is not a difficult task. We just need to find a subsequence of symbols in the measure that add up to the time signature. And by scoring these paths, we can pick the best scoring paths and build the possible edge set from them. But there is one problem with this method. One or several symbols might be left out from the monophonic paths we choose. In that case, the possible edge set will not contain any edges that connect to these symbols. We do not allow this to happen since each symbol must be connected to the rhythm graph.

To solve this problem, we pick monophonic paths in the following way. For each symbol, we find the t best scoring monophonic paths that go through this symbol. For example, in Figure 5.4, we find four most likely monophonic paths that pass through the circled note, and we label the edge that *goes into* and *goes out of* the note as red. From the figure, we can see that there are two possible *goes into* edges and two *goes out of* edges in these. Thus we include these four red edges into the possible edge set. We

repeat this process for each symbol in the measure and build the possible voice edge set. In this way, we make sure that each symbol will have at least one edge that connects to other symbols.



(a) (b)

(c) (d)

Figure 5.4: Four different monophonic paths that pass through the circled note.

Now given that there are many monophonic paths that go through a certain symbol, how can we pick t best scoring paths? We will apply dynamic programming to solve this problem and the algorithm will be introduced in Section 5.3.

5.3 Algorithm for the monophonic path selection

As discussed in Section 5.2, our goal is to find the top t best scoring monophonic paths that go through symbol v_k. We consider all rhythm bearing symbols in a system

measure as vertices, which we denote as $V = \{v_0, \dots, v_n\}$. V is an ordered sequence where vertices are sorted from left to right according to their location in the measure. We define a vertex variable $x_v = (x_v^o, x_v^d)$ to represent the onset and the duration state of a symbol. Duration x_v^d could be either the nominal duration or triplet duration. Since the vertices are sorted, when we put a voice edge between two vertices, the only direction that the voice can go is from v_i to v_j, where $i < j$. And their values satisfy the constraint that $x_{v_j}^o = x_{v_i}^o + x_{v_i}^d$.

To find a monophonic path that goes through certain symbol v_k, we are actually looking for a *subsequence* of V, which we denote as $V_{mono} = \{V_{i_1}, V_{i_2}, \dots V_{i_k}\}$, such that $i_1 < i_2 < \cdots < i_k$ and $\sum_{v_j \in V_{mono}} x_{v_j}^d = T$. The voice edges over the subsequence, V_{mono}, are denoted as $E_{mono} = \{e_{v_j, v_{j+1}} | v_j \in V_{mono}\}$.

We define $H(x)$ as the score over a certain monophonic path as:

$$H(x) = \sum_{v \in V_{mono}} G_v(x_v) + \sum_{e \in E_{mono}} G_e(x_e)$$

where the edge score function $G_e(x_e)$ tells the plausibility that two vertices are connected by a voice edge, and $G_v(x_v)$ scores a vertex variable. These two scoring functions are similar to the scoring function in the polyphonic graph-based algorithm.

Now the problem is converted to an optimization problem where we want to find the subsequence V_{mono} that maximizes the value of $H(x)$ under the constraint that $v_k \in V_{mono}$ and $\sum_{v_j \in V_{mono}} x_{v_j}^d = T$.

5.4 Optimization with dynamic programming

We use dynamic programming to solve the optimization problem. Assuming symbol v_k must exist in the path, we work on the symbols $x_j \in V$ from left to right in order. The state of the left barline is $x_0 = (0,0)$ and the state of the right bar line is $x_n = (T, 0)$.

The most common way of solving the problem is as follows. We define $\widehat{H}\left(x_{v_j}\right)$ as the best score of monophonic path that stop on v_j, under the constraint that v_j is labeled as x_{v_j}. Then we can write $\widehat{H}\left(x_{v_j}\right)$ with the following recursive equation:

$$\widehat{H}\left(x_{v_j}\right) = \begin{cases} \max\limits_{\substack{v_i: i<j \\ x_{v_j}=x^o_{v_i}+x^d_{v_i}}} \left(\widehat{H}(x_{v_i}) + G_e\left(e_{v_i,v_j}\right) + G_v\left(x_{v_j}\right)\right) & \text{for } j \le k \\ \max\limits_{\substack{v_i: k\le i<j \\ x_{v_j}=x^o_{v_i}+x^d_{v_i}}} \left(\widehat{H}(x_{v_i}) + G_e\left(e_{v_i,v_j}\right) + G_v\left(x_{v_j}\right)\right) & \text{for } j > k \end{cases}$$

We compute $\widehat{H}\left(x_{v_j}\right)$ recursively by solving the larger problem with the solution of smaller problem for $1 \le j \le n$, and finally we obtain a *single* best monophonic path.

However, the best-scoring path is not necessarily the correct monophonic path. For example, if a correct path jumps between staves or jumps between symbols with different stem directions, it will receive a large penalty on the scoring function and get pruned. Therefore, we want to keep the diversity of v_k by including different states of v_k in dynamic programming as well.

The following is our algorithm.

Step 1:

First, we apply the same dynamic programming procedure as described above from the left barline untill we get to symbol v_k. Thus we get the best paths arriving on different possible states of v_k. We have

$$\widehat{H}_{fwd}(x_{v_0}) = 0$$

$$\widehat{H}_{fwd}\left(x_{v_j}\right) = \max_{\substack{v_i : i<j \\ x_{v_j} = x^o_{v_i} + x^d_{v_i}}} \left(\widehat{H}_{fwd}(x_{v_i}) + G_e\left(e_{v_i,v_j}\right) + G_v\left(x_{v_j}\right)\right), \qquad j = 1,2,\dots,k-1$$

Step 2:

Now we have different possible states of $\widehat{H}_{fwd}(x_{v_k})$ and we want a path to reach the right barline so that the rhythm is consistent with the time signature. We still use a similar dynamic programming method, but this time we move backwards from the last symbol, the right barline, to symbol v_k. We define the $\widehat{H}_{bwd}\left(x_{v_j}\right)$ as the best score of the monophonic path that goes from the right bar line and stops on v_j, under the constraint that v_j is labeled as x_{v_j}. We have

$$\widehat{H}_{bwd}(x_{v_n}) = 0$$

$$\widehat{H}_{bwd}\left(x_{v_j}\right) = \max_{\substack{v_l : l>j \\ x_{v_j} = x^o_{v_l} + x^d_{v_l}}} \left(\widehat{H}_{bwd}(x_l) + G_e\left(e_{v_j,v_l}\right) + G_v\left(x_{v_j}\right)\right), j = n-1, n-2, \dots, k$$

Step 3:

Now that we have paths from both directions, we can just join the two paths together if they lead to the same state x_{v_k}. The score of the whole path is:

$$\widehat{H}(x_{v_k}) = \widehat{H}_{fwd}(x_{v_k}) + \widehat{H}_{bwd}(x_{v_k})$$

With the three steps above, we keep the best scoring path that passes through symbol v_k with different states x_{v_k}, and we also make sure that the monophonic paths start on $(0,0)$ and ends on $(T, 0)$. Finally, we trace back and reconstruct the path and retrieve the voice edges that connect to v_k. The edges that directly connect to symbol v_k will be included in the possible edge set. We do this process on each symbol v_k in the measure to abtain the possible edge set.

Compared to the heuristic method discussed, there are two advantages with the monophonic path method. First, it looks over all symbols for a successful voice path that is consistent with the time signature, which is more robust than making local decisions, especially on notation violation cases where the heuristic function does not apply.

Second, the correct monophonic path will also propose the correct (onset, duration) state in the rhythm graph. In other words, if a (onset, duration) state does not exist in any monophonic paths, it probably does not exist in the polyphonic graph either. For example, for the circled note in Figure 5.4, the monophonic path in (a), (b), (c) gives the (onset, duration) pair as (5/16, 1/16) (which is the correct state), and (d) gives the (onset, duration) pair as (3/8, 1/16). They provide two possible states for the circled

note. In the graph-based algorithm, if the state generated for this symbol is not equal to either of the two states, it is probably wrong and we can remove it from our search space. It helps us reduce the computation in our polyphonic graph-based algorithm.

6 Human-in-the-loop correction

The graph-based rhythm algorithm discussed so far is fully automated. It takes the recognized symbols and interprets the rhythm without human interference. However, like all other AI systems, we couldn't expect it to work correctly 100% of the time. Especially with a challenging problem such as OMR, there are always rare examples that violate the notation rules that lead to failure cases in OMR systems. Therefore, we need some way to correct results when the automatic algorithm fails.

Most OMR commercial systems have a music notation editor that allows users to correct errors, i.e. inserting missing symbols, deleting wrong symbols, labeling missing triplets. This method can be laborious work and separates the recognition and correction procedure.

In this chapter, we propose a human-in-the-loop system to allow users and the system to work together to correct errors. It is part of our interactive OMR system, Ceres, which allows correction in both recognition and interpretation errors. We will focus on the interpretation part in this thesis. The biggest difference between our system and other commercial systems' correction editors is that our system incorporates user input in the process of computation, taking users' input as constraints and re-computing the result iteratively until they achieve the desired result. Users can provide two forms of input: labeling onset and duration for a certain symbol, or changing parameters of models. Considering the rhythmic relations between symbols in a measure, we hope that by labeling one symbol, the system will automatically correct several errors.

6.1 Correction of commercial OMR systems

Almost all commercial OMR systems, such as SmartScore [4] or SharpEye [19], have a music notation editor that allows users to correct errors. Figure 6.1 shows an interface of the editor for SmartScore. The system shows the recognition results below the original score image. It also highlights the measures in pink to remind users that there could be some recognition errors (mostly when the rhythm is not consistent with the time signature).

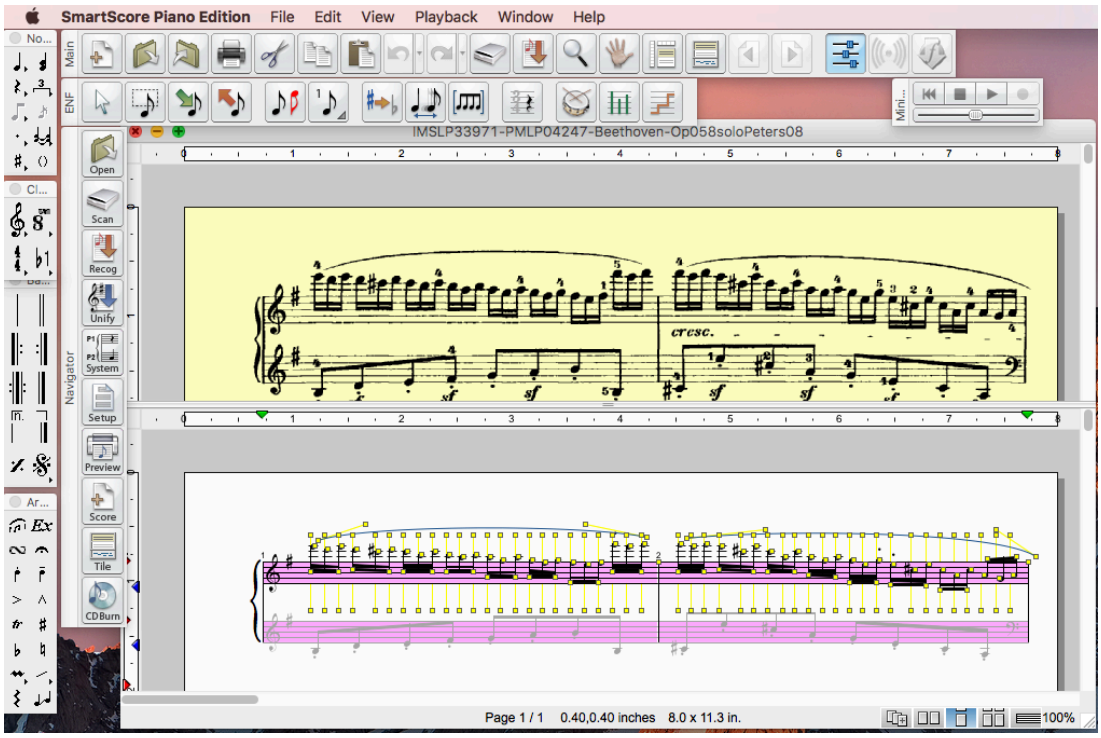


Figure 6.1: The editor interface for SmartScore. The editor allows the user to correct errors.

There are two types of errors: recognition errors and interpretation errors. A recognition error refers to an error that happens in the recognition step, for example an eighth note recognized as a quarter note, or a missing rest. On the other hand, an

interpretation error refers to a case where a symbol is recognized correctly, but the system fails to understand the correct music meaning.

Correcting interpretation errors can be messy. We have discussed the example of interpretation error in SmartScore, as shown in Figure 6.2. Two circled notes are labeled with the wrong voice and it leads to the wrong duration of the red voice. In order to fix this example, the user needs to label the circled notes as the third voice or even insert omitted rests to make a voice complete. This correction process could be very time consuming and laborious work when the whole page is full of errors.

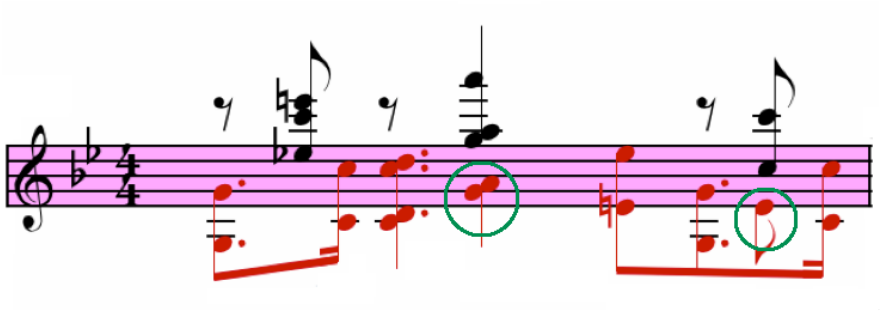


Figure 6.2: An example of missing 8-tuplet and missing 9-tuplet. The time signature is in 4/4.

6.2 A human-in-the-loop system

Here we propose a human-in-the-loop correction method, which is part of our OMR system, Ceres. Our previous work [78], [79] introduced correcting recognition errors. In this chapter, we focus on correcting interpretation errors.

6.2.1 Diagram of the system

The process of human-in-the-loop recognition is shown in the diagram in Figure 6.3. The interactive process can be described as follows:

(1) The system first interprets the rhythm of the measure and presents users the result by showing the rhythm graph and playing back the music.

(2) Users correct the symbols on the measures where there are errors.

(3) The system takes users' input as constraints and re-interprets the rhythm.

(4) Repeat this process until users produce the correct rhythm.

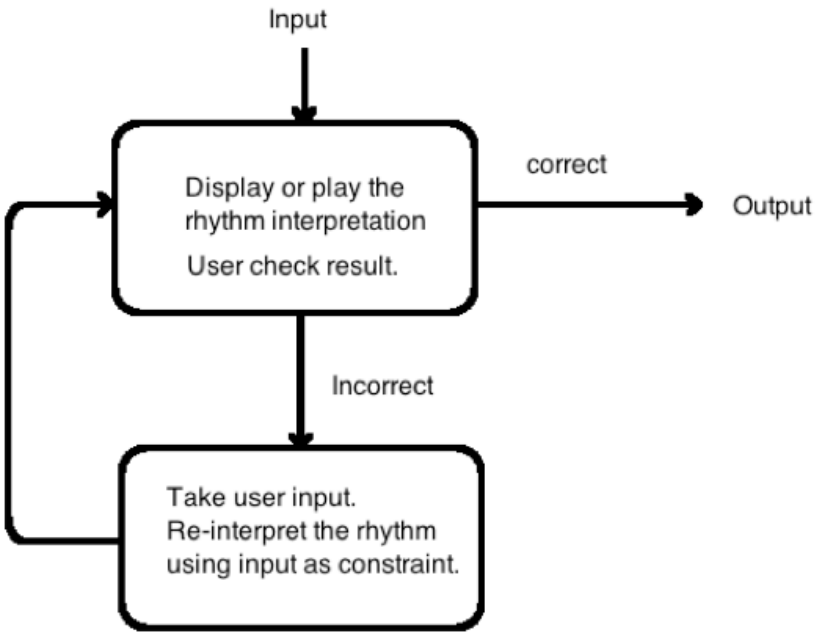


Figure 6.3: Diagram for the human-in-the-loop rhythm interpretation system.

Rhythm interpretation sometimes can be difficult to check, especially when there are missing tuplets. The most common way to check the correctness of rhythm is by listening to the music. We convert the result of rhythm and pitch to MIDI events and

allow users to play back the music. However, listening to all measures one after another might take quite some time. We also visualize the result by plotting the rhythm graph on the score to allow users to check the rhythm.

In our human-in-the-loop system, users can give two types of input: (1) label a symbol's onset and duration; or (2) change the parameters in our model. We will explain them in the following two sections.

6.2.2 Labeling a symbol's onset and duration

Labeling rhythm can be somewhat tricky. The method used by SmartScore is to label voices, but the voices can be subjective, especially when they merge or split in polyphonic music. Different people may have different understandings of the voices.

To avoid this ambiguity, we choose to directly label the onset and duration of a given symbol, as shown in the interface of our user input system in Figure 6.4. Users can click on a certain symbol and type in the rational numbers of duration and onset. There are two advantages with this form of input. The first is that the onset and duration of a symbol are almost always unique, which avoids the ambiguity of labeling voices.

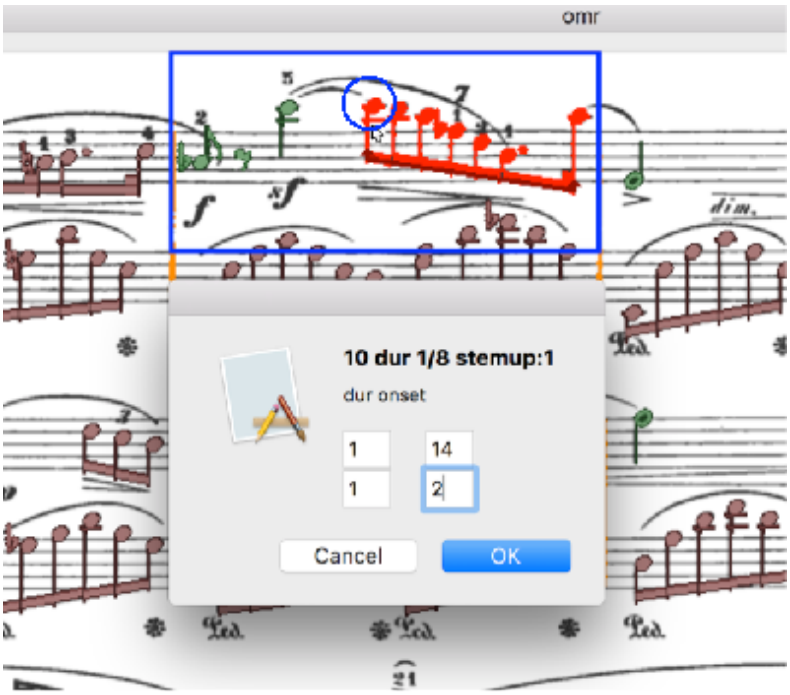


Figure 6.4: User interface to label the onset and duration of a given note. In this example, the user clicks on the first note of the 7-tuplet note and labels its onset as 1/2 and duration as 1/14.

The second advantage is that the (onset, duration) pair can be easily built as constraints into our dynamic programming algorithm. More specifically, we can remove the states that are inconsistent with user labels in the following way.

Assuming that the vertices in the graph are $V = \{v_1, v_2, \dots, v_{n-1}, v_n\}$, users can label one or more vertices in the measure. We denote users' input as $U_{Input} = \{x_{v_i} = x^*_{v_i} | v_i \in V\}$, which means vertex v_i must be labeled as $x^*_{v_i}$. The objective function of the rhythm graph is defined as:

$$G(x) = \sum_{v \in V} G_v(x_v) + \sum_{e \in E} G_e(x_e)$$

To remove the states that are not consistent with users' input, we can just set the

score on a vertex to $G_v(x_{v_i}) = -\infty$ when $x_{v_i} \neq x^*_{v_i}$. This will ensure that we only keep states that are consistent with users' input in the algorithm.

We use users' inputs as constraints in the dynamic programming process of the monophonic path, as well as in building the rhythm graph. Because symbols are all connected in our graph-based algorithm, the constraint of one symbol can influence other symbols through the coincidence or voice edges. Therefore, our system could correct several errors at once by using the connection between the symbols, thus minimizing the user's correcting work.

6.2.3 Changing parameters

Another form of user input is to change parameters in our model. In Chapter 1, we showed an interpretation error of not being able to identify missing triplet, as shown in Figure 6.5. To fix the error in SmartScore, one needs to label each beam group as triplets explicitly one by one, which requires a lot of work. Instead, our goal is to the system once that there are missing triplets, and let the system figure out the rest.

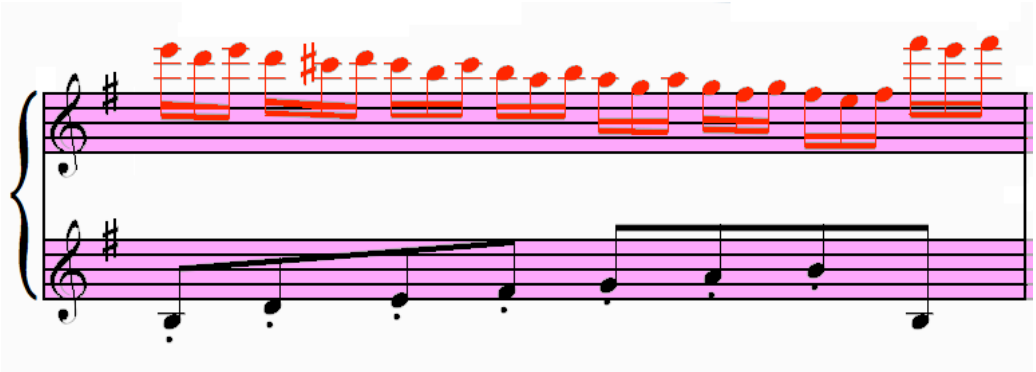


Figure 6.5: An example of an interpretation error of missing triplets in SmartScore.

Therefore, we propose allowing the user to change the parameters in our model.

This input primarily tells the systems which notation rules are allowed and which are not. Our interface has two check boxes: (1) Allow missing triplet. (2) Allow voice jumping between staves. By switching the states of these check boxes, users can change the parameters used in the score function in our model. For example, checking the "allow missing triplet" box will decrease the penalty on labeling symbols as triplet notes. Similarly, checking the "allow voice jumping between staves" box will increase the score of a voice edge that connects two notes belonging to two staves.

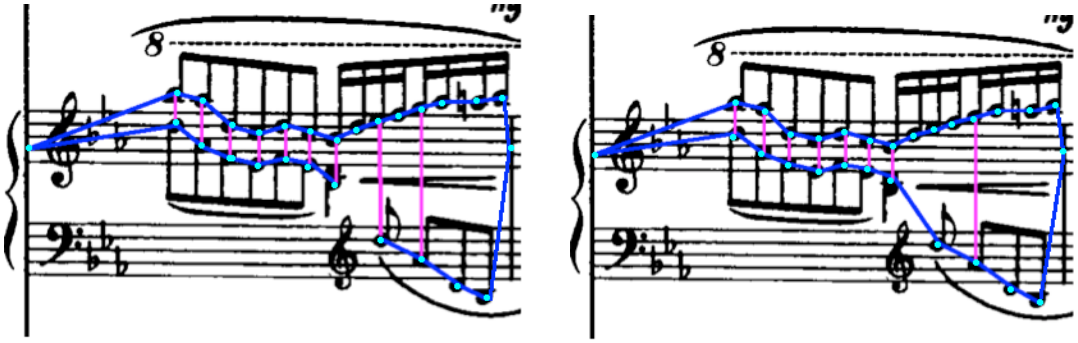
The reason we allow changing these two parameters is because these are the most important rules for rhythm interpretation, yet are also frequently violated. The default parameters penalize labeling notes as missing triplets and voice jumping between staves to avoid really complex interpretation, which works with simpler measures (which compose the major portion of most pieces), but might in cases where there are many missing triplets or notation rules are violated frequently in the measure. On the other hand, when the parameters are set to be "tolerant" with complex interpretations, it leads to more mistakes on simpler examples.

Therefore, in order to be able to handle the rare complex examples without interfering with the majority of simple cases, users should always choose the most stringent combination of model parameters.

6.3 Correction examples

In the following section, we give two examples where our system makes the wrong interpretation and how they are fixed with user input.

6.3.1 Example 1



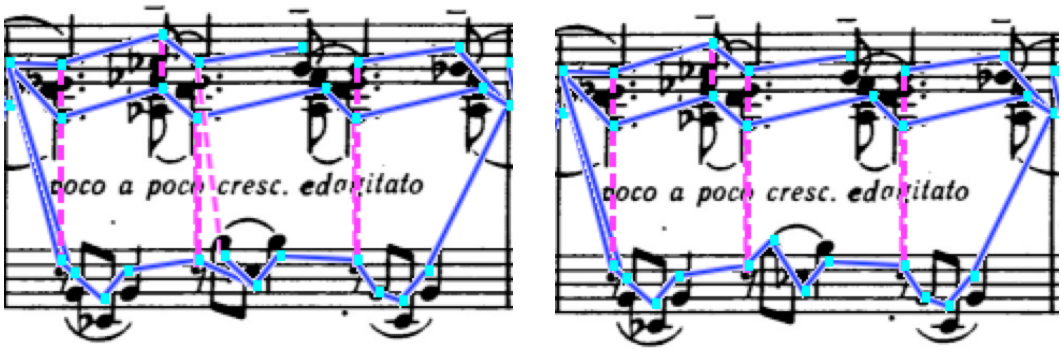
(a) (b)

Figure 6.6: An example from Concerto No.2 by Rachmaninoff. Time signature is in 4/4. (a) shows the rhythm graph generated by the system. (b) shows the correct rhythm graph.

Figure 6.6 shows an example of two rhythm graphs, where (a) is the wrong rhythm graph generated by our system and (b) is the corrected rhythm graph. From (a), we see that the system labeled the quarter note in the upper staff and first 8th note in the lower staff as normal notes while they are actually missing triplets. The states of these two notes generated by the system are (1/2, 1/4) and (5/8, 1/8), while the correct states are (1/2,1/6) and (2/3,1/12).

Users can fix this error by labeling the state of the quarter note as (1/2,1/6). With this constraint on the quarter note, the algorithm will remove the states in (a) since it contradicts the users' label. As the result, rhythm graph (b) becomes the best scoring result and two errors were fixed with one correction.

6.3.2 Example 2



(a) (b)

Figure 6.7: Example from Sonata No.1 by Rachmaninoff. Time signature is in 3/2. (a) shows the rhythm graph generated by the system. (b) shows the correct rhythm graph.

Figure 6.7 shows another example of a wrong rhythm graph in (a) and corrected graph in (b). In the lower staff in this measure, there are three groups of missing triplet that are consisted of an 8th rest and two beamed 8th notes. However, our system interpreted the second group of triplet notes as normal 8th note. The 8th rest and the first 8th note in the beamed group are so close that they seem to the same onset time and the 8th note is shifted right to avoid overlapping. The extra coincidence edge in (a) indicates this interpretation.

To fix this problem, users can label the onset of the left note of the beamed 8th note as (7/12,1/12). By using this label as a constraint, our algorithm correctly labels all three 8th notes as triplets and thus estimates the correct rhythm for the whole measure.

6.4 Conclusion

In this section, we presented a human-in-the-loop rhythm interpretation system. Instead of using conventional notation editors, we take input from users and use it as constraints to re-compute the rhythm interpretation. Users can either directly label the onset and duration or change the model parameters. Our goal for this design is to minimize the human's work in correcting errors.

7 Experiments

To evaluate the graph-based algorithm proposed in this work, we performed an experiment of rhythm interpretation on a data set of 50 pages of scores. We constructed a dataset specifically for the rhythm interpretation problem on piano scores, hoping to cover different types of challenges encountered in rhythm interpretation.

The organization of this chapter is as follows. In Section 7.1, we give a general introduction of our dataset. We will discuss how and why we chose these pages. In Section 7.2, we discuss how we evaluated the rhythm interpretation result with a quantitive metric. The evaluation of OMR is a challenge in itself, and there is no universal evaluation metric for rhythmic accuracy. We discuss several options and explain why we chose the one we used in our evaluation task. In Section 7.3, we present our experimental result, and finally in Section 7.4, we summarize challenges in the dataset into four categories. We discuss each category with examples and present the rhythm graph generated by our system.

7.1 Dataset

The performance of an OMR system could be significantly influenced by different features of a data set (i.e. the image quality, the music font, and the notation difficulties). Different datasets evaluate different aspects of a system. There is no standard dataset for OMR, especially for rhythm interpretation tasks. Byrd [52] proposed a small dataset for evaluating OMR system, but it is more for recognition tasks instead of interpretation

and is too simple for rhythm interpretation. Therefore, we built a dataset specifically for the rhythm interpretation task. We picked 50 pages of score images from 24 pieces for our experiments.

The dataset we created is focused on piano scores for the following reasons. First, piano music contains the most challenging examples. Many composers were famous pianists and they composed a large collection of rhythmically challenging piano pieces. With the goal of building a digital music library, we cannot avoid the most central problems. Second, since the piano is played with two hands, piano music provides more polyphonic music notations than other monophonic instruments do, which provides more examples of polyphonic music. Finally, focusing on piano scores helps focus our problem in a more consistent framework, rather than sprawling on different notation problems for different instruments like guitars, percussions, etc.

All scores are downloaded as PDF files from the IMSLP website [80]. The lengths of the scores we picked range from 8 to 60 pages. We selected between 1 to 5 pages from each score rather than doing experiments on the whole, because we want our datasets to contain challenging and distinct examples. We observe, for example, that in many pieces, the most challenging measures in many pieces appear only in a small section of the whole piece, while the rhythms for the rest of the pages are very simple. Thus experimenting on the whole piece will be unnecessary. In other cases, some pieces may contain difficult rhythms throughout the whole piece. However, they are mostly repeated rhythms. In such cases, experiments on the similar rhythm fail to produce new information.

A summary of the dataset is presented in Table 7.1. We include different composers, meters, fonts, publishers and most importantly different notation patterns. Although 50 pages is still a small-scale test set, we try to make it cover the most typical and distinct challenges one would encounter in piano scores.

Table 7.1: Dataset

Piece Name	Composer	Publisher
The 6 Partitas	J.S.Bach	Leipzig: C.F. Peters, No.205, n.d. Plate 9792.
Sonata No.13 in B-flat major	Wolfgang Mozart	Leipzig: Breitkopf & Härtel, 1878. Plate W.A.M. 333.
Sonata No.23	Ludwig Van Beethoven	Leipzig: Breitkopf und Härtel, 1862-90. Plate B.146.
Concerto No.4	Ludwig Van Beethoven	Leipzig: C.F. Peters, n.d.(ca.1897). Plate 8603
Sonata No.13 in A major	Franz Schubert	Leipzig: Breitkopf & Härtel, 1888. Plate F.S. 102.
Sonata No.21 D.960	Franz Schubert	Leipzig: Breitkopf & Härtel, 1888. Plate F.S. 107.
Fantaisie Impromptu	Frédéric Chopin	Leipzig: C.F. Peters, n.d.[1879]. Plate 6216.
Polonaise Phantaisie	Frédéric Chopin	Leipzig: Breitkopf und Härtel, n.d.[1878]. Plate C. V. 7.
Liebesträume	Franz Liszt	Leipzig: Fr. Kistner, n.d.(1903). Plate K.E. 31
Das Rheigngold (for piano solo)	Richard Wagner	Milan: G. Ricordi & C., n.d. Plate 53816.
Die Walküre (for piano solo)	Richard Wagner	Mainz: B. Schott's Söhne, n.d. Plate 25346.
6 Klavierstücke Op.118	Johannes Brahms	Leipzig: C.F. Peters, No.3300b, n.d.(ca.1910). Plate 9488.
Variations on an Original Theme	Johannes Brahms	Leipzig: C.F. Peters, No.3300a, n.d.(ca.1910). Plate 9487.
Islamey	Mily Balakirev	Hamburg: D. Rahter, 1902. Plate 2897
Sonata in C-sharp Minor	Pyotr Tchaikovsky	Moscow: Muzgiz, 1945. Plate M. 18919
Sonata G major	Pyotr Tchaikovsky	Leipzig: Edition Peters, 1960. Plate E.P.12017.
2 Arabesques	Claude Debussy	Paris: Durand & Fils, 1904. Plate D. & F. 4395-96.
Suite Bergamasque	Claude Debussy	Leipzig: Edition Peters, 1969. Plate E.P. 12420b.
Préludes (book1)	Claude Debussy	Paris: Durand et Cie., 1910. Plate D. & F. 7687.

Piece Name	Composer	Publisher
Sonata No.1	Sergei Rachmaninoff	Moscow: Muzgiz, n.d.(1948). Plate M. 18453
Le tombeau de Couperin (piano)	Maurice Ravel	Paris: Durand & Cie., 1918. Plate D. & F. 9569.
Sonata No.7	Sergei Prokofiev	Moscow: Muzgiz, 1955. Plate M. 23818

7.2 Evaluation metric

For our evaluation, we choose the representation of the pair of (onset, duration) relative to the measure. We argue that the onset and duration are sufficient for evaluating the rhythm, because rhythm is, after all, about when to play a note and for how long to play it. The biggest advantage of this approach is that the onset and duration are objective.

An alternative representation of onset is through voices. In fact, musicXML [81], one of the most well known symbolic representations, represents rhythm with voices. Every note in the system measure is assigned to a voice. The notes in one voice are played one after another. If voices split or merge, musicXML uses *forward* and *backup* elements to move the note to the correct musical time. This method can represent any rhythm. The problem is that there is no unique way to label a voice. Therefore, there could be several correct voice labels that correspond to the same rhythm, which makes it ambiguous to create a unique, correct ground truth.

We construct the ground truth by manually labeling each symbol with a pair of (onset, duration). The values of onset and duration are all rational numbers in the range of the time signature. The duration of a symbol is labeled as its true duration. For

example, a missing triplet eighth note's duration will be 1/12. Almost all rhythms are unambiguous to interpret for evaluated measures. We evaluate the rhythm recognition accuracy on one page of the score. We compute the error rate on a page by:

$$Error\ rate = 1 - \frac{N}{M}$$

where M is the total number of duration bearing symbols on a page and N is the number of symbols interpreted with both the correct onset and duration.

7.3 Discussion of results

The accuracy for 50 pages is listed in the Table 7.2. We computed two error rates, the error rate with a uniform setting and the error rate with the optimal setting on one page. For the uniform setting, we used a single set of parameters on all 50 pages. This is the most *tolerant* setting, that is, it allows missing triplet and voices to jump between staves with small penalties on the score function. For the optimal setting on a single page, on the other hand, we used the parameter value, based on the user's setting on each page. For example, if there are no missing triplets on the page, the user likely chose to penalize more on triplet labeling. Similarly, if there are no voices jumping between two staves, we also penalize more on that. This avoids complex interpretation and also greatly reduces the search space of our algorithm.

We achieved an accuracy of 90% with the uniform setting and 95% on an optimal setting on each page. We can see that the error rate on the optimal single page setting is lower than the one with the uniform setting. This can be seen as a bias-

variance trade-off. The uniform parameter setting would be able to produce complex interpretations, but will cause more errors in some simpler pages that do not have any missing triplet or voice jumps between staves.

In the following sections, we discuss the success and failure cases in different notation challenges. We group different rhythm notation challenges into four categories. In each category, we show examples and plot the rhythm graph generated by our algorithm.

Table 7.2: Evaluation result.

	Piece Name	Composer	Page	Meter	uniform	optimal
1	The 6 Partitas	J.S.Bach	21	3/2	0.17	0.00
2			39	3/4	0.11	0.06
3			69	3/4	0.01	0.00
4	Sonata No.13 in	Wolfgang	1	4/4	0.03	0.00
5	B-flat major	Mozart	7	4/4	0.06	0.01
6	Sonata No.23	Ludwig Van	1	12/8	0.09	0.01
7		Beethoven	7	12/8	0.28	0.00
8	Concerto No.4	Ludwig Van	8	4/4	0.11	0.11
9		Beethoven	11	4/4	0.07	0.03
10	Sonata No.13 in	Franz	1	4/4	0.02	0.02
11	A major	Schubert	9	6/8	0.02	0.00
12	Sonata No.21	Franz	18	3/4	0.09	0.09
13	D.960	Schubert	19	3/4	0.02	0.02
14	Fantaisie	Frédéric	1	4/4	0.02	0.02
15	Impromptu	Chopin	4	4/4	0.06	0.06
16			5	4/4	0.01	0.01
17	Polonaise	Frédéric	2	3/4	0.06	0.00
18	Phantaisie	Chopin	11	3/4	0.09	0.05
19	Liebesträume	Franz	1	4/4	0.05	0.00
20		Listz	5	4/4	0.04	0.04
21			9	3/4	0.07	0.00
22			10	3/4	0.11	0.11
23	Das Rheigngold	Richard	28	9/8	0.15	0.08
24	(for piano solo)	Wagner	33	9/8	0.07	0.01
25			43	3/4	0.28	0.18
26			48	4/4	0.03	0.07
27	Die Walküre	Richard	6	3/2	0.04	0.04
28	(for piano solo)	Wagner	8	3/4	0.04	0.05
29	6 Klavierstücke	Johannes	1	4/4	0.04	0.04
30	Op.118	Brahms	2	4/4	0.03	0.03
31			3	3/4	0.01	0.01
32	Variations on an Original Theme	Johannes Brahms	4	3/8	0.04	0.04

	Piece	Composer	Page	Meter	uniform	optimal
33	Islamey	Mily Balakirev	4	12/16	0.05	0.05
34	Sonata in C-sharp	Pyotr	21	3/4	0.00	0.00
35	Minor	Tchaikovsky	25	3/4	0.29	0.29
36	Sonata G major	Pyotr Tchaikovsky	44	2/4	0.28	0.28
37	2 Arabesques	Claude	1	4/4	0.02	0.02
38		Debussy	2	4/4	0.01	0.01
39	Suite	Claude	1	4/4	0.00	0.00
40	Bergamasque	Debussy	6	3/4	0.25	0.01
41			13	9/8	0.15	0.01
42	Préludes	Claude	2	4/4	0.03	0.03
43	(book1)	Debussy	37	3/8	0.26	0.07
44	Sonata No.1	Sergei	3	4/4	0.00	0.00
45		Rachmaninoff	13	3/2	0.05	0.05
46	Sonata No.2	Sergei Rachmaninoff	1	4/4	0.25	0.25
47	Le tombeau de	Maurice	7	6/8	0.01	0.01
48	Couperin (piano)	Ravel	9	4/4	0.00	0.00
49	Sonata No.7	Sergei	21	3/4	0.05	0.05
50		Prokofiev	28	3/4	0.02	0.01

7.4 Examples of rhythm graphs

In this section, we discuss four categories of challenges in rhythm interpretation. We show each one with examples from our data set and show the rhythm graph generated by the system.

7.4.1 Voice splitting/merging and jumping between staves

One feature of our graph-based method is that it abandons the idea of looking for a complete *voice*, but only cares about the *voice edge* and *coincidence edge* that connect notes with their neighboring notes.

The advantage of our method is that it allows much more flexibility in the cases where voices split and merge and jump between staves or beams. We do not assume that voices must exist through the whole measure, and nor do we need to define a maximal number of voices. In this section, we show some examples to demonstrate this advantage.

(a) Splitting and merging voices

Figure 7.1 shows a polyphonic measure with splitting and merging cases. The upper staff measure seems to have two voices, one with stem up direction and the other with stem down direction. However, the two circled notes are extra notes that do not belong to either voice. Assigning voices to these two notes will be tricky since they do not belong to any complete voice.

Our algorithm finds the correct rhythm for this case. From the rhythm graph, we can see that our method naturally captures the splitting and merging voices by finding the predecessor for these two notes.

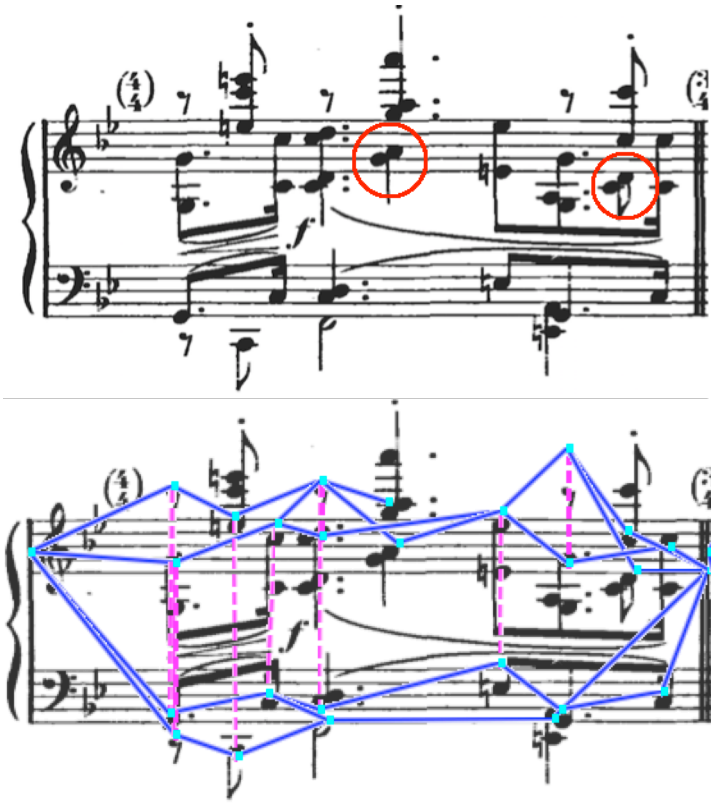


Figure 7.1: Example from the Préludes (book 1) by Claude Debussy. Time signature is in 4/4.

Figure 7.2 shows a similar voice splitting and merging example. In this example, there are at least four voices throughout the measure. Looking for complete voices would be especially difficult in this example. Our algorithm successfully interpreted the rhythm. From the rhythm graph, we can see that the algorithm made use of all vertical alignment between the stem up and stem down voices.

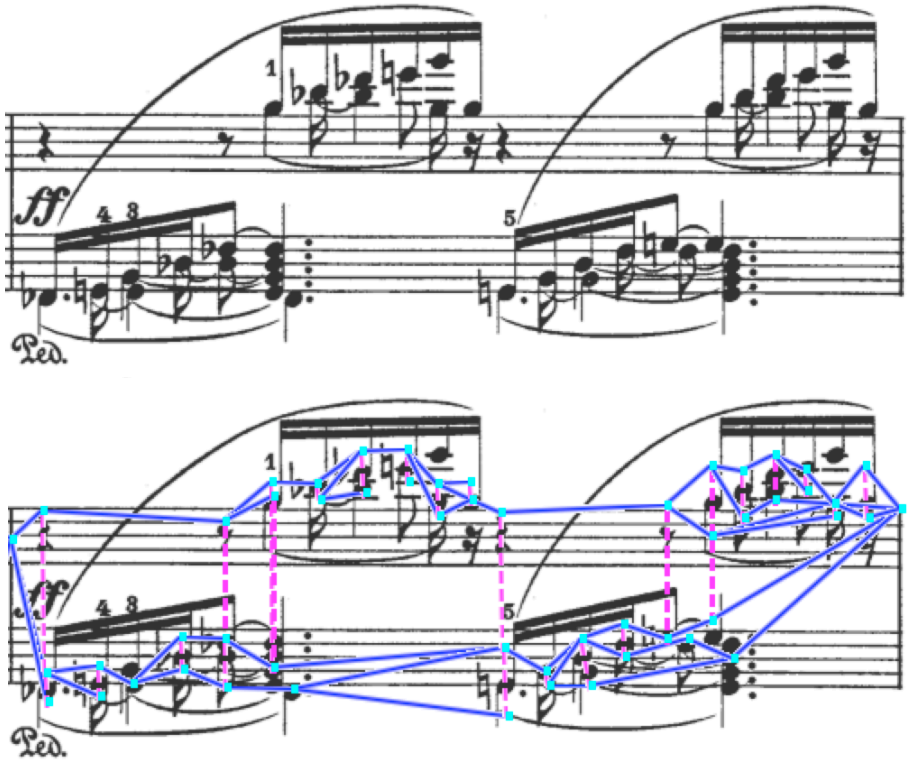


Figure 7.2: Example from piano sonata op57 by Ludwig Van Beethoven. Time signature is in 12/8.

(b) Voices jumping between staves

We build the rhythm graph on the system staff (or grand staff) measure and allow putting voice edges between notes that belong to different staves. This is because it is common for voices to jump between staves in piano scores. If we build a rhythm graph within a staff measure, we will very likely fail to find a complete voice.

Figure 7.3 shows an example by Beethoven. This is a challenging example because it consists of numerous beamed and isolated missing triplets, and the upper voice begins at the bass clef and then jumps to the treble clef.

Our algorithm successfully interpreted this measure. Our monophonic path algorithm also plays an important role in it as it always tries to find complete monophonic paths even though it involves one lower score edge.

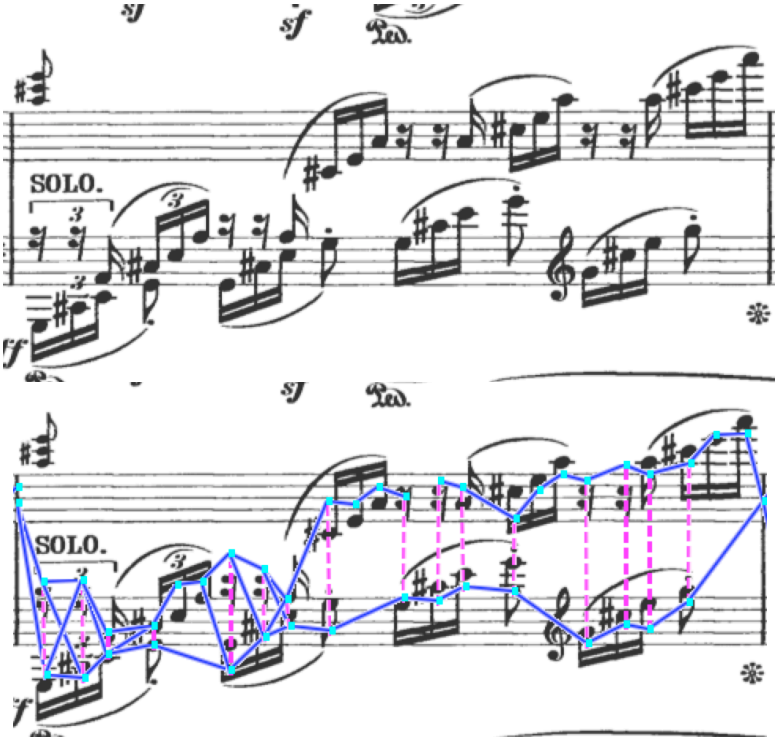
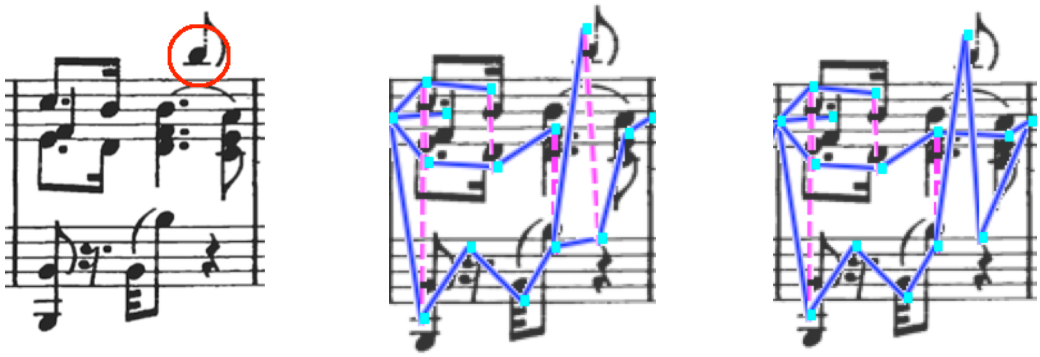


Figure 7.3: Example from Piano Sonata No.4 by Beethoven. Time signature is in 4/4.

Figure 7.4 shows an example from Piano Sonata D.960 by Franz Schubert. In this example, the challenge is that the circled note in the upper voice actually belongs to the voice in the lower staff. The correct voice is shown in the right figure; the voice on the lower staff jumps to the upper staff and then jumps back to the lower staff.

Our algorithm did not estimate the correct rhythm in this measure. The generated rhythm graph is shown in the middle figure. One reason for this confusion is that the voice jumping back and forth will cause two voice edges to have a lower score.

The other reason is that the 8th note in Figure 7.4 (B) and the quarter note in the bass clef almost align, which gives a good score on the coincidence edge. Therefore, the middle interpretation has a better score and gets picked by our algorithm.



(a) (b) (c)

Figure 7.4: An example from Sonata in B flat D.960 by Franz Schubert. Time signature is in 3/4. Figure (b) shows the rhythm graph generated by the system. Figure (c) shows the correct rhythm graph.

(c) Voice jumping between beamed groups.

Another feature of our model is that we treat each stem in a beam as an individual vertex rather than treating a whole beamed group as a vertex. We also always allow putting voice edges between two beam notes. The disadvantage of this decision is that we have many more vertices and possible edges, which makes computation much more complex. The advantage is that it can deal with cases where voices jump between beamed notes.

In figure 7.5, the bass clef staff seems to have two voices. The upper voice consists of stem up notes and the lower voice consists of stem-down notes. However neither of the voices has enough duration of 12/16. All notes actually belong to a single voice, and the voice jumps between beamed notes with different stem directions.

Our algorithm interprets the correct rhythm for this example. The challenge here is that the voice edges jumping between different beams have low scores. One main reason that our algorithm can still produce the correct rhythm is that the treble clef staff has notes that vertically align with the symbol in the bass clef, which provides four high score coincidence edge. Thus, globally this rhythm graph gets the highest score.

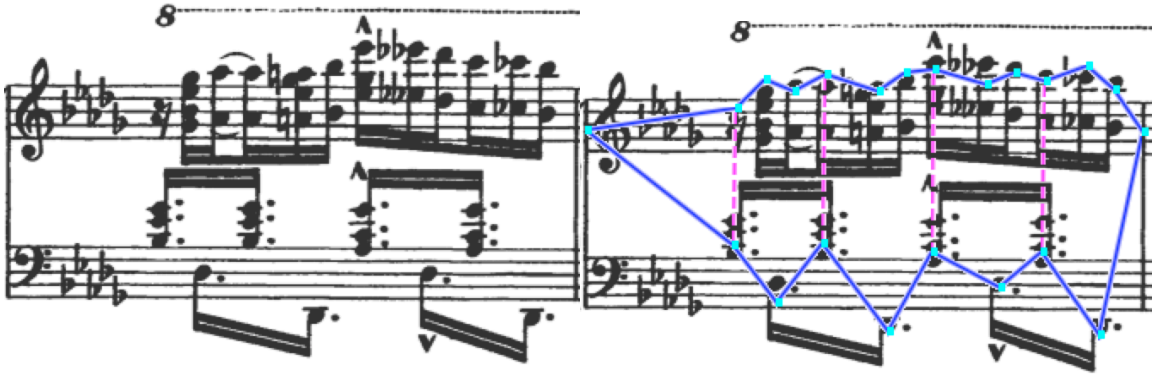


Figure 7.5: Example from Islamey by Mily Balakirev. Both measures are in time signature 12/16.

Figure 7.6 shows another example from the Préludes by Debussy. In the measure, the voice jumps between two beamed notes in two staves. Another challenge in this example is that there is no vertical alignment like the one in the previous example.

Our algorithm interprets the correct rhythm. One main reason is that although these edges have bad scores, these edges are the only voice edges generated by our monophonic path algorithm. The monophonic path algorithm not only generates the correct possible edge set, but also pruned wrong edges that do not lead to successful voice paths.

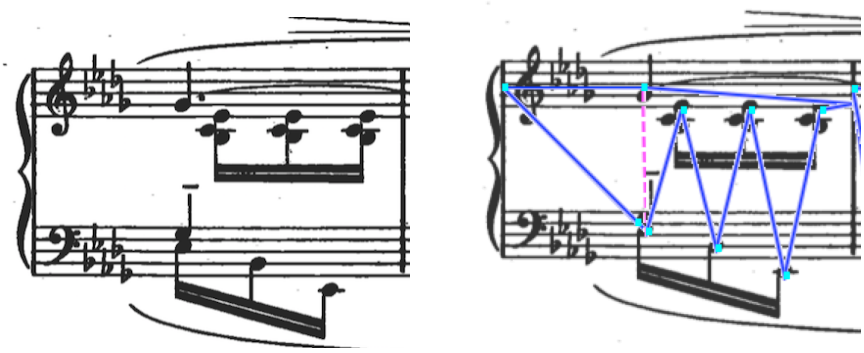


Figure 7.6: An example from the Préludes (book1) by Claude Debussy. Both measures are in time signature 3/8.

7.4.2 Misalignment and false alignment

Vertical alignment provides very important evidence in checking consistent rhythm interpretation for polyphonic music. However, in many examples, notes that are supposed to be aligned can often be shifted to avoid overlapping, thus lead into misalignment between notes that have the same onset time, or even false alignment between notes that do not have the same onset time.

The distance between two shifted notes can vary quite a lot. Thus rule-based method does not work. We cannot pick a threshold and decide whether two notes should have the same onset time just based on the horizontal distance. Instead, we rely on the global optimization to find the best interpretation. In this section, we present some ambiguous alignment examples and show that our method is robust to deal with these cases.

(a) Misaligned notes

Figure 7.7 shows an example of misaligned notes in the upper staff. The first note of the stem up voice and the first note of the stem down voice are played at the same onset

time and their pitches differ by half a note. To avoid overlapping of two note heads, one note head is shifted by the distance of a note head. In this example, the coincident notes have *aligned stems*, that is, two stems of two voices are aligned vertically, which is a very common notation in polyphonic music.

As shown in the rhythm graph, our algorithm correctly interprets the rhythm and places the coincidence edge on the shifted coincidence notes.

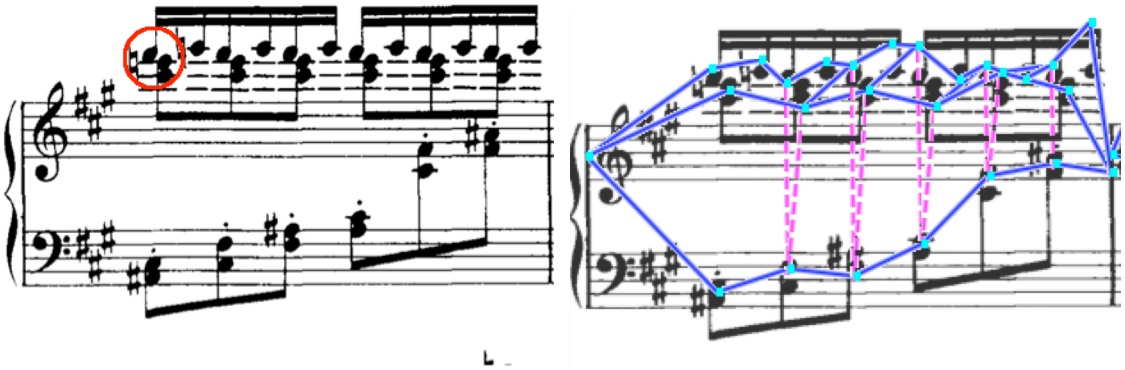


Figure 7.7: Example from Sonata in A major by Franz Schubert. Time signature is in 6/8.

However, not all coincident notes have *aligned stems*, and not all *aligned stems* are coincident notes. Figure 7.8 shows an example where the upper measure has a mixture of duplet and triplet eighth notes. As the score gets crowded, one cannot decide whether its coincidence just depends on how close two notes are, or whether stems are aligned or not. In the left bounding box, two notes have the same onset time but two stems are not aligned, while in the second bounding box, two notes seem to have aligned stems but do not have the same onset time. This example shows that there is no consistent rule-based pattern for shifted notes.

Our algorithm correctly labeled all missing triplet and interpreted the correct rhythm again. The global optimization makes sure that our algorithm does not make decisions just based on one observation such as vertical alignment, but considers the whole interpretation.

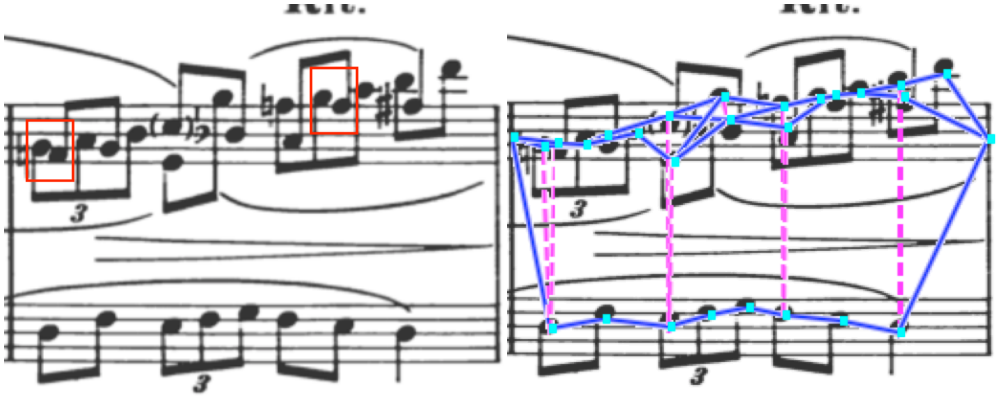


Figure 7.8: Example from Le tombeau de Couperin for piano by Ravel. Time signature is in 4/4.

(b) Falsely aligned notes

We just discussed cases where notes played at the same time might not align. Now we discuss the opposite direction, that notes that are aligned vertically might not have the same onset time.

In the upper staff in Figure 7.9, the quarter note in F seems to align with the second eighth note in the lower staff. However, it has the same time as the half note but is shifted right by larger distance because of the wrong-sided note on the left.

Our algorithm correctly interprets the rhythm. From the rhythm graph, we can see that the voice edges connect the notes in the correct order and get the globally optimized interpretation.

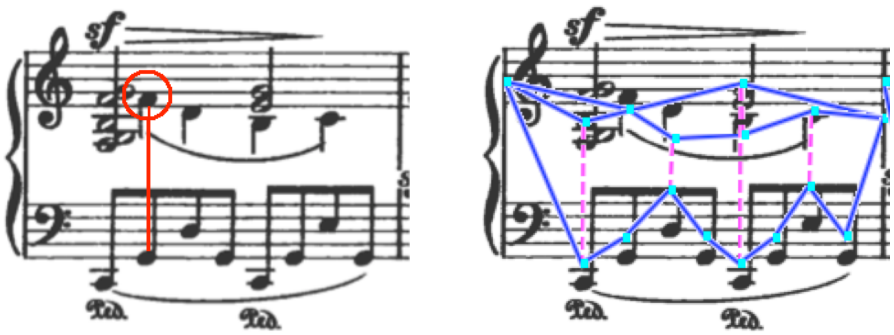


Figure 7.9: Example from Six Pieces for Piano by Johannes Brahms. Time signature is in 4/4.

Misalignment also happens quite often when there are tuplet notes. Figure 7.10 shows an example where the notes in the upper staff are 16th notes and the notes in the lower staff are triplet 8th notes. Ideally, if these notes are placed proportionally according to their duration, they should not align except for the ones on the down beats. However, we can see from the figure that at least two pairs of notes appear to be exactly aligned.

If our algorithm decides coincidence edges just based vertical alignement, it would choose these false coincidence edges and interpret the rhythm incorrectly. Instead, it generates the correct rhythm graph with global optimization.

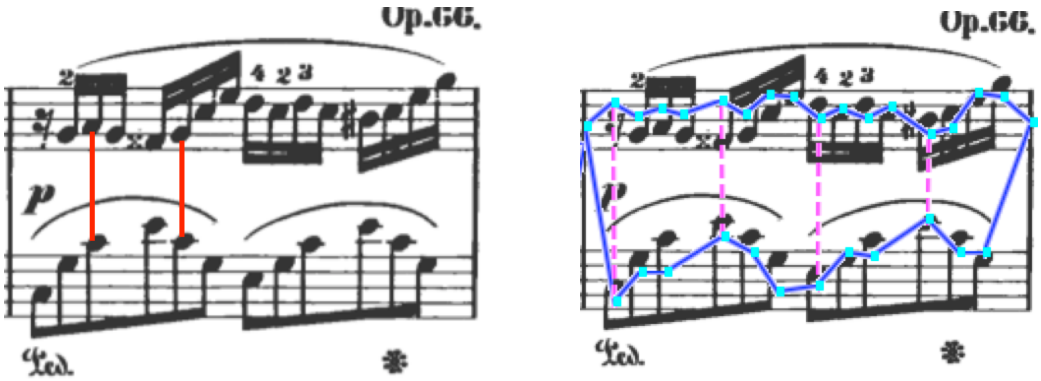


Figure 7.10: Example from Fantaisie Impromptu op66 by Frédéric Chopin. Time signature is in 4/4.

Figure 7.11 shows another example of the same problem of false alignment, where the upper measure has a 7-tuplet and the lower measure has a 6-tuplet. Apparently, the 7-tuplet is not placed proportionally. We can see two pairs of false alignment between the 7-tuplet and the 6-tuplet. Again, our algorithm correctly interprets the rhythm in this measure.

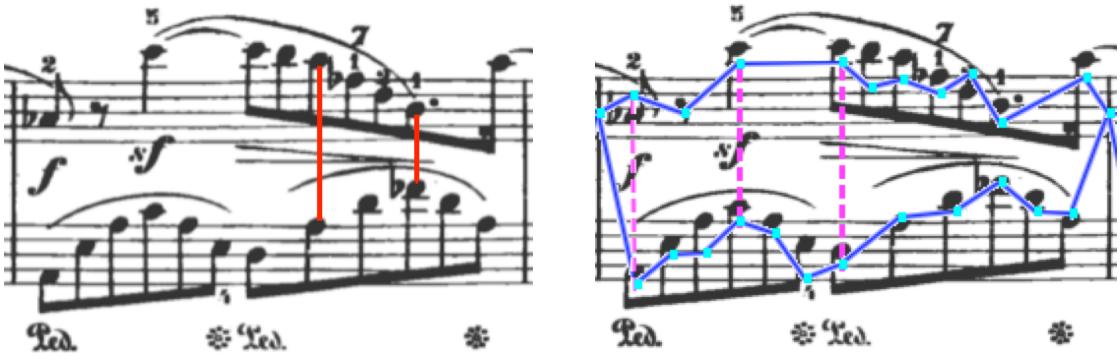


Figure 7.11: Example from Fantaisie Impromptu op.66 by Frédéric Chopin. Time signature is in 4/4.

7.4.3 Missing tuplets

In the previous two sections, we showed the challenging examples of identifying polyphony voices. In this section, we will show examples of missing tuplets. We will first focus on missing triplet since they are the most common type of missing tuplet and will then generalize to other types of tuplet later.

Triplet notes come in groups of three, i.e. three 16th notes or three 8th notes. But as we looked through more piano scores, we found that they can be a combination of different lengths of notes, rests, beamed notes and isolated notes. One beamed note can also be a partial triplet and a partial non-triplet. The problem that comes with so

many variations of missing triplets is that we cannot enumerate all of the patterns and find the pattern in the measure.

The key idea is that we deal with this problem by individually labeling each note as triplet or non-triplet. The advantage is that this covers any combination of the triplet labeling. In the following section, we will pick some examples to show the different variant types of missing triplets and the interpretation result of our system.

(a) Missing triplet

Figure 7.12 shows an example of a missing triplet in three beamed 16^{th} notes. The difference is that the triplets are only part of a beamed note. The first case is two groups of missing triplet 16^{th} notes in one beamed note. The second case is a group of missing triplet beamed with a non-triplet eighth note.

As shown in the rhythm graph, our algorithm correctly labels all the missing triplets in this measure by making use of the vertical alignments between voices. In this example, five groups of 16th notes are missing triplets, which leads to a large triplet note penalty on the score. However, the vertical alignment compensates the triplet label penalties and leads to the correct interpretation.

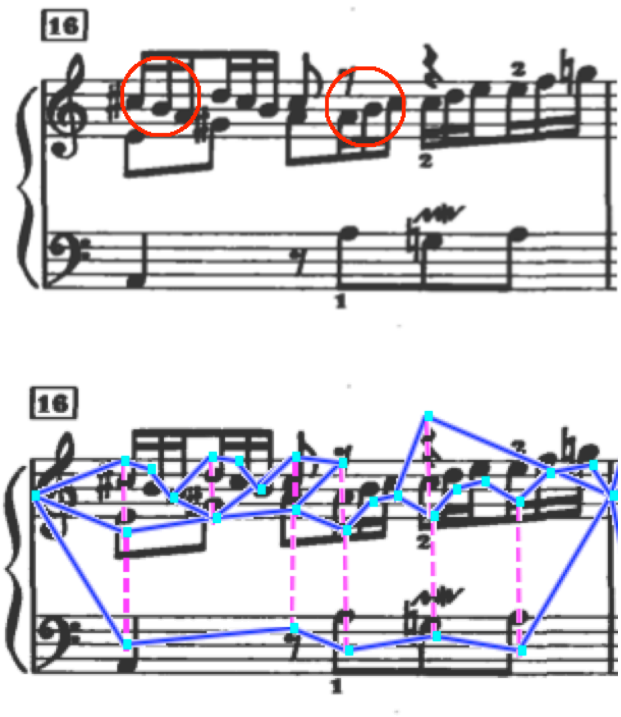


Figure 7.12: Example from Partita No.3 by J.S. Bach. Time signature is in 3/4.

While a beamed note can be partially triplet and partially non-triplet, separate symbols can also form a group of triplet notes. Figure 7.13 shows a typical example of separate missing triplets, where two 16th rests and an isolated 16th note form a group of missing triplets.

Our algorithm correctly labeled all the triplet 16th notes in this measure. It is worth mentioning that both voices are both missing triplets in parallel. When triplets appear in parallel, labeling them as either triplet or non-triplet will not cause a contradiction immediately. The major contradiction in this example comes from the time signature constraint. Incorrectly labeling the triplet notes as non-triplet notes will cause voices to end longer than the time signature.

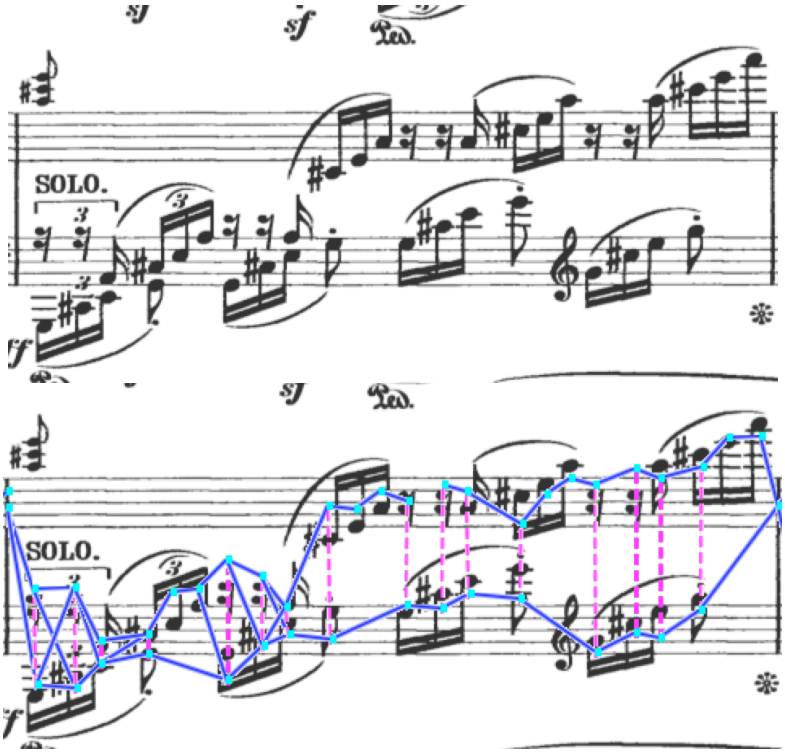


Figure 7.13: Example from concerto No.4 by Beethoven. Time signature is in 4/4.

Labeling triplets individually is also useful for dealing with missing 6-tuplets and 9-tuplets. Figure 7.14 shows an example of a missing 9-tuplet. The 9-tuplet consists of an eighth rest and a beamed note consisting of eight 8th notes. The rhythm interpretation is similar to triplets, where 9-tuplet notes taking the duration of 3/4 can also be seen as three groups of triplets each taking a duration of 1/4. Thus we use the same method to solve this example.

As shown in the rhythm graph, our algorithm correctly labeled the 9-tupelts by making use of the coincidence edge. However, it makes a mistake of labeling the quarter rest and eighth note in D natural as triplet as well. The correct interpretation should be that the 8th note follows the dotted quarter note in C in the lower staff. This is an

interesting example showing that even our algorithm can make local errors in some notes, but the interpretation of most notes are still reasonable.

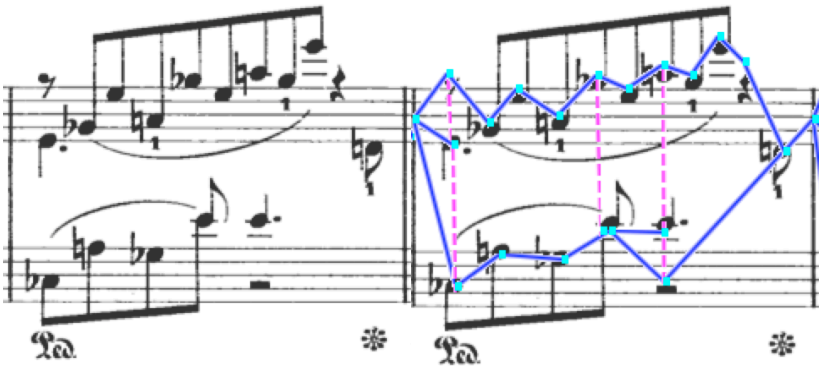


Figure 7.14: An example from Liebestraume by Franz Liszt. Time signature is in 4/4.

Figure 7.15 shows a more complex example in which a triplet note can be a combination of beamed notes and isolated notes. This measure has both missing triplets and 6-tuplets. Similarly, the 6-tuplet can also be interpreted as two groups of triplet notes. This is also an example where voices jump in and out of beamed notes.

Our algorithm correctly interpreted this example by making use of the voice edge and coincidence edges. This example shows that by labeling triplet notes individually, our method is flexible enough to cover complex combinations of triplet notes.

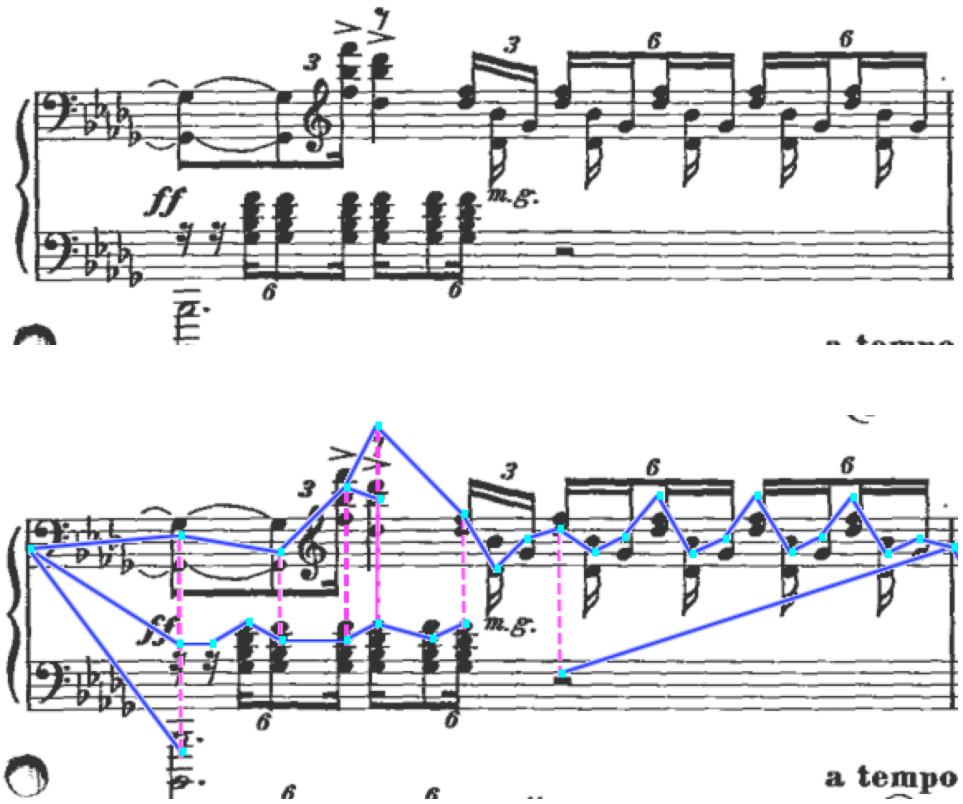


Figure 7.15: Example from Sonata No.2 by Sergei Rachmaninov. Time signature is in 4/4.

(b) Missing duplet

Having shown some missing triplet examples, now we move to the missing duplets. Duplets are a group of two notes that fit into the duration of three non-duplet notes. They often appear in meters that have groups of three notes, such as 3/8, 6/8, 9/8, etc. While they appear frequently in a piece of score, they can also be omitted and become missing duplets.

The same idea to solve missing triplets can be generalized to duplet notes naturally. We just consider two hypotheses of the nominal duration and the duplet duration.

Since duplets and triplets occur in a different time signature, we do not consider both triplet and duplet hypotheses at the same time, but choose one or the other according to the time signature.

Figure 7.16 shows an example of duplet. The time signature in this measure is 12/8. In the upper staff, the beamed note of two 8th notes is a duplet note and each eighth note takes a duration of 3/16.

Similar to missing triplets, we rely on the vertical alignment between voices to identify the missing tuplets. When two voices are duplet in parallel, we also rely on the meter constraint because mistakenly labeling the duplet note as a non-duplet will cause voices to end sooner than the time signature.

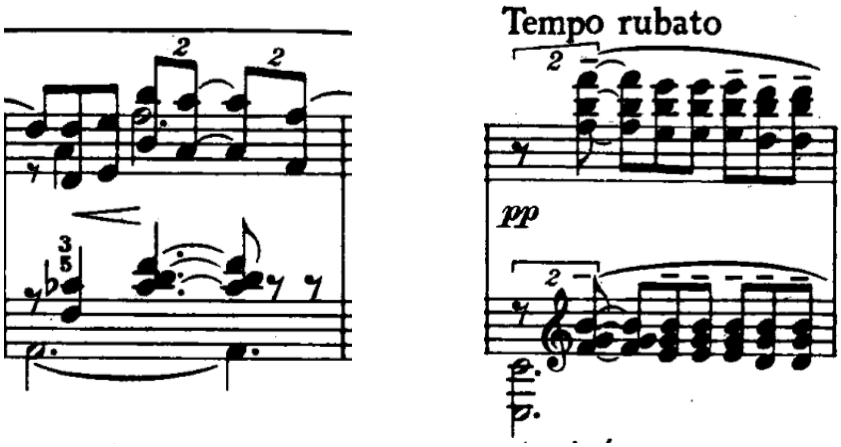


Figure 7.16: An example of duplets. Time signature in 9/8.

Figure 7.17 shows an example of a 4-tuplets in a measure. There are two beamed 4-tuplets; the one in the lower staff is labeled and the one in the upper staff is omitted. Just like 6-tuplet can be seen as two pairs of triplets, 4-tuplets can also be seen as two

pairs of duplets. Thus we interpret missing 4-tuplets with the same method as missing duplets.

Our algorithm correctly identifies the missing 4-tuplet in this example. It makes use of the vertical alignment and the meter constraint to correctly interpret the rhythm of the measure.

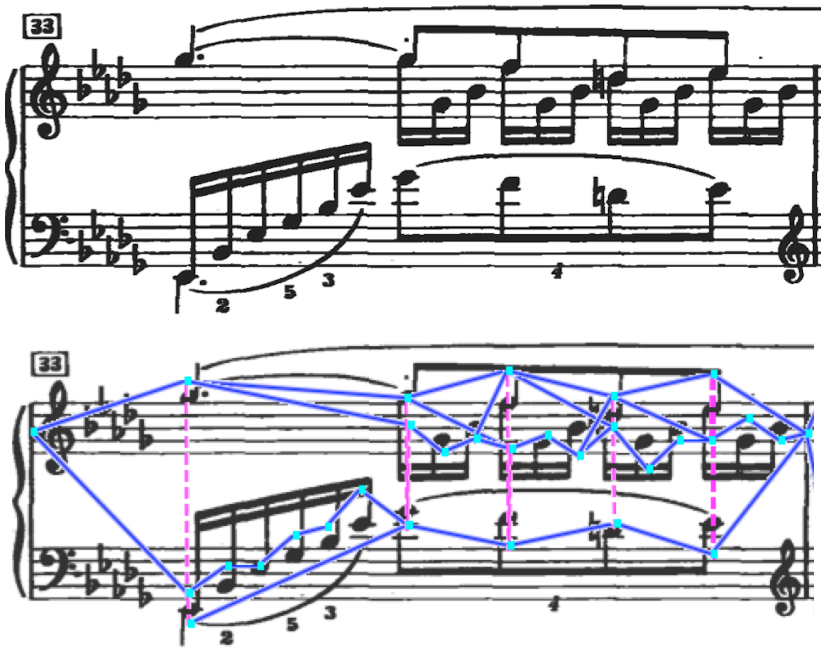


Figure 7.17: An examples from Bergamasque by Claude Debussy. Time signature is in 9/8.

7.4.4 Shared note heads and impossible rhythm

In this section, we discuss the notation of *shared note heads* and a related notation called *impossible rhythm*. Shared note head is another common notation in piano music. It happens when two notes belonging to two voices are overlapped on the score because they have the same pitch and onset time.

(a) Shared note heads

The two notes sharing the note head should have the same onset time but can have different durations. It is trickier to identify the durations of both note heads when the shared note head has an augmentation dot. The augmentation dot can belong to one or both of the note heads. Thus identifying which note head has the augmentation dot is the key problem. Our graph-based model naturally handles the problem by allowing multiple duration hypotheses when labeling the vertices. Each note head has both hypotheses of with and without augmentation dots.

More specifically, we solve the problem with the following method. We treat notes that share the same note head as two individual vertices in the rhythm graph, typically one stem up note and one stem down note. We do so because (1) it is easier to assign them with different durations, and (2) we can use them independently in monophonic voice paths. As for the constraint that notes sharing a note head should have the same onset time, it can also be built naturally as a special case of vertical alignment. Since sharing a note head leads to the same horizontal location, the coincidence edge will provide a positive score that strongly encourages them to have the same onset time.

Figure 7.18 shows a typical example where a note head is shared by a stem-up 16th note and a stem down quarter note. In this case, the augmentation dot only belongs to the stemmed down quarter note. The symbol recognition step would be able to recognize that the dot belongs to the shared note head but cannot identify which note has the dot without understanding the rhythm.

Thus in our algorithm, we consider two duration hypotheses for the note head, which are with and without augmentation dot for both symbols. And by building the rhythm graph, we found that the stemmed-up beamed sixteenth note forms a complete voice that adds up to the time signature, and so are three stemmed down dotted eighth note. Thus we get the correct the durations for both symbols.

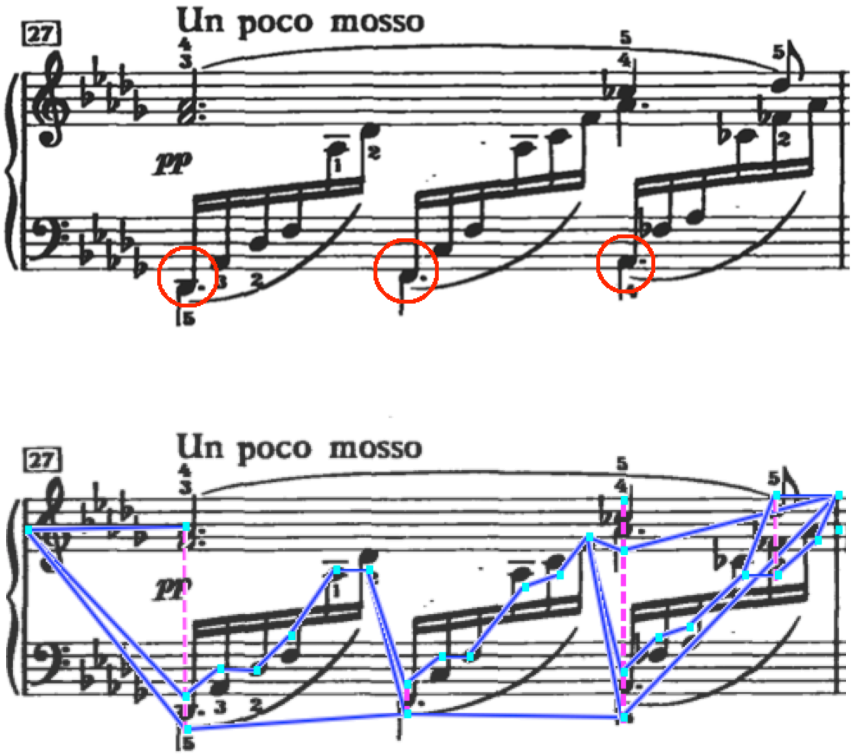


Figure 7.18: An: example from Bergamasque by Claude Debussy. Time signature is in 9/8.

Figure 7.19 shows an example where the augmentation dot on the shared note head is omitted. In this example, a stem down 8th note and a stem up 8th note share the same note head without an augmentation dot. If we take a closer look at the stem up note, the other notes on the same stem have an augmentation dot. Since note heads that belonging to the same stem should have the same number of augmentation dots, the

shared note head belonging the stem up note should also have augmentation dot, but it is omitted to match with the stem down eighth note.

Our system deals with the case in the following way. Our recognition step has a constraint that note heads on one stem must have the same type of augmentation dots. Thus the recognition result adds the omitted augmentation dot on the shared note head. Then we solve the problem in the same way as discussed in the previous example by considering two duration hypotheses for both symbols. From the rhythm graph, we can see that the algorithm interprets the rhythm correctly by labeling the stem up notes as normal dotted rhythms and stemmed down notes as 8th triplet notes.

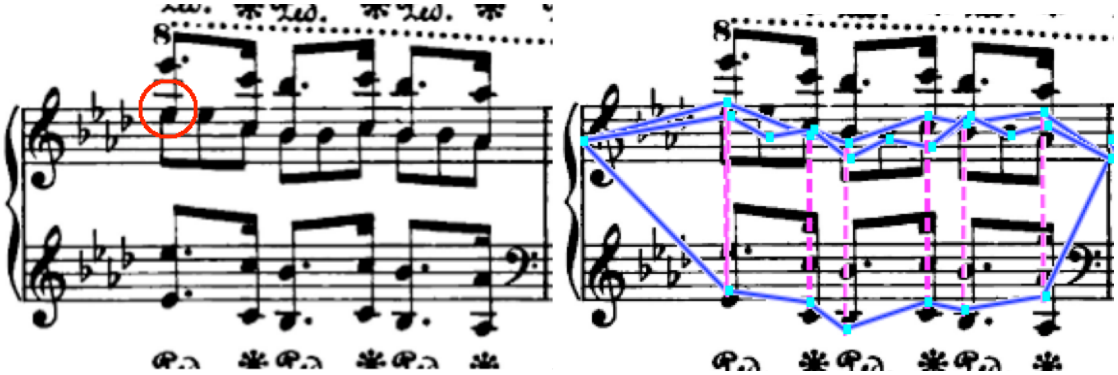


Figure 7.19: An example from Polonaise Phantaisie op61 by Frederic Chopin. Time signature is in 3/4.

Figure 7.20 shows an example where notes with different head types can still share a note head. In this example, the stem down half note and the stem up eighth note share the same note head. When a solid note head overlaps with an open note head, we can see that the open note head is drawn on the score. If we use the closed note head, people will not be able to understand that the stemmed down note is a half note.

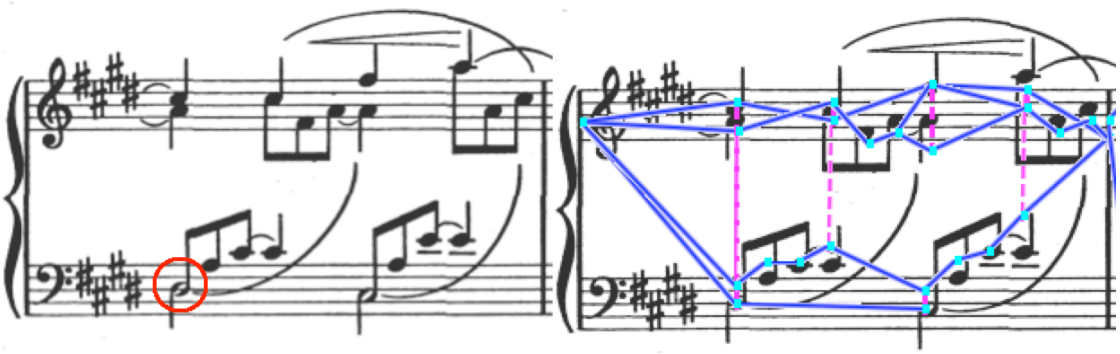


Figure 7.20: An example from Arabesque by Claude Debussy. Time signature is in 4/4.

(b) Impossible rhythms

We just discussed shared note heads notation, and now move on to a related notation, *impossible rhythm*. Figure 7.21 shows a typical example of impossible rhythm.

There are multiple ways to interpret the impossible rhythm. Here we will just consider two options, the triplet prior or the duplet prior interpretation. In this example, triplet priority plays the stem up notes in triplet time and moves the stem down notes to match with the onset of triplet rhythm. In duplet priority, the duplet note will be played strictly in tuple time while the first two 16th notes of triplet voice will be played as a 32nd note so that the third 16th note matches with the onset of the duplet note. This interpretation follows the theory that the tuplet voice has a coherent melody.

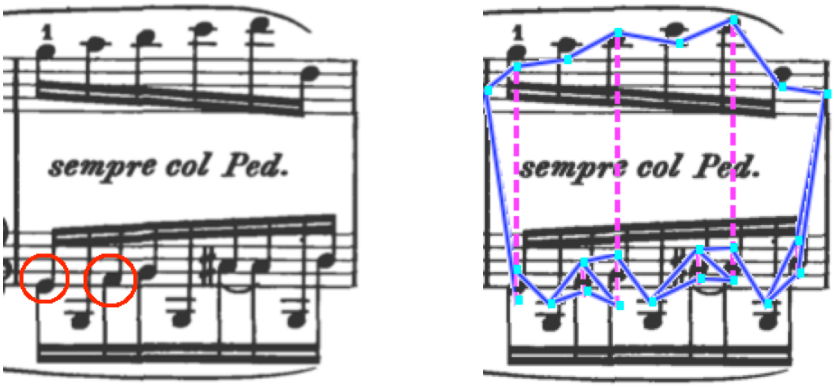


Figure 7.21: An example from Variations on an Original Theme by Johannes Brahms. Time signature is in 3/8.

The challenge of this problem is that when we choose one beamed note as priority rhythm, the notes in the other beamed note are no longer played consecutively. Our model can deal with this situation because we do not have a constraint that neighboring notes must be connected by a voice edge. From the rhythm graph, we can see that our algorithm gives the triplet priority interpretation.

Figure 7.22 shows another example of impossible rhythm. In this example, the stem up note is a normal beamed note, while the stem down beamed eighth notes are in triplet rhythm. The contradiction of onset also happens to the second shared note head, where the stem up voice leads to an onset of 3/16 on the second shared note head and the stem down voice leads to an onset of 1/6.

This notation raises the controversial question, whether the doted notes should match the triplet rhythm or not. Scholars have no strict answer to this question. One

interpretation that prior on the triplet rhythm is called triplet assimilation, and the other interpretation will prior on the dotted rhythm. [56]

The way our algorithm interprets the rhythm of this measure is that it just plays the stem-up voice in non-triplet dotted rhythm and plays the stem-down voice in triplet rhythm. This does lead to a contradiction on onset time on the second shared note for each pairs of beam groups.

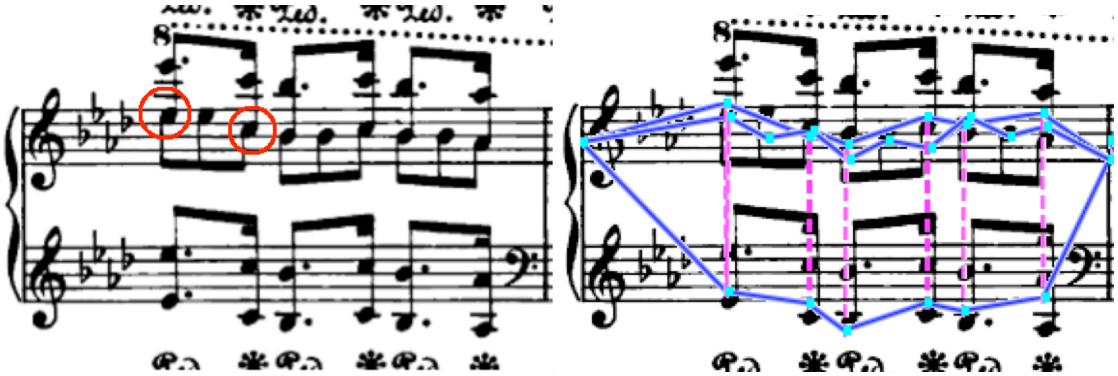


Figure 7.22: An example from Polonaise Phantaisie op61 by Frederic Chopin. Time signature is in 3/4.

To sum up in this chapter, we showed many challenging examples of music notation measures in our experimental dataset. The reason we chose these cases is that we wanted to see how our algorithm performs when notation rules are violated or the notes miss information on tuplet labels. From our results, we show that our graph-based algorithm is generally capable of handling these cases with the globally optimal interpretation.

8 Summaries and conclusions

In this thesis, we studied rhythm interpretation in an OMR system, and we focused on polyphonic piano scores. The two major challenges of rhythm interpretation are (1) multiple voices mixed in one measure splitting and merging through the measure, and (2) durations are ambiguous due to missing triplet. We proposed a rhythm graph model where we treat music symbols as vertices and represent sequential/alignment rhythm relationships as edges. The rhythm graph can capture the complexity and variety of the rhythm structure and also handle the ambiguity of duration labels. We formulate the problem as a global optimization that maximizes the score of a legal rhythm graph.

The major contribution of this work is that we studied how to interpret rhythm in polyphonic scores, especially when the notations are ambiguous or violated. We looked through more than 100 piano scores and studied the most difficult examples. We discussed several important notation rules and the scenarios in which they are violated in Chapter 2. We notice a heavy tail effect that many notations that violate the notation rules may repeatedly appear in one score or a certain composer's work. This means that if we make hard rule-based decisions based on the common notation rules, our system will certainly fail to interpret these examples. Instead, our model scores the plausibility of a certain interpretation based on all perspectives and looks for the optimal solution over all legal configurations.

In Chapters 3 and 4, we introduced the details of our graph-based method. The method finds the optimal configuration by looking for the best labels on the symbol vertices and the possible edges. The graph made by all vertices and possible edges is

often a generalized graph with loops, so we cannot directly apply the max-sum algorithm to perform inference. Instead, we convert it into a clique tree and then apply dynamic programming to find the optimal labeling. The computation complexity is exponential to the size of the biggest clique.

The choice of the possible edge set can make a big difference on the performance of our algorithm. Missing the correct edge in the possible edge set will lead to failure to build the correct rhythm graph, while too many false edges will cause a large clique size and slow down the computation. In Chapter 5, we presented a monophonic path method to improve our algorithm by finding more reliable possible edge sets. The monophonic path method is a fast pre-processing step that looks for the successful monophonic path in the polyphonic measure. The successful monophonic path provides more reliable possible vertices and edge label hypothesis and significantly reduces the search space in our later algorithm by ruling out impossible states.

The graph-based algorithm with the monophonic path method together finds the optimal interpretation. However, in real life, the optimal interpretation is not always the correct one. In Chapter 6, we presented a human-in-the-loop algorithm that allows users to correct interpretation errors. The difference between our interactive correction method and commercial OMR systems is that it takes human input and uses that as a constraint to re-compute the interpretation. Since the rhythms of connected symbols connected are highly correlated, fixing one error could potentially fix all errors in a measure, which saves user's manual correction as much as possible.

Finally, we presented experimental results on a data set of 50 pages of scores, specifically built to cover the most common rhythmic challenges in piano scores. We showed the generated rhythm graphs and discussed the successful and failure examples.

The biggest contribution of this work is that we explored how to apply measure level music notation knowledge in rhythm interpretation. Results show that our graph-based method is robust enough to deal with cases where notation rules are violated and triplet symbols are omitted. Because we formulate the problem as a global optimization, another advantage of our system is that errors usually occur in a small part of a measure, instead of interfering with the whole measure.

However, the problem of rhythm interpretation is far from being solved. One limitation of our model-based method is that it can only solve cases where the true duration is considered in the model's hypotheses and when the most basic rules (i.e. the meter rule, the connected graph) are obeyed. But there are always exceptions that do not follow our models, such as a missing 17-tuplet, Cadenza, or certain impossible rhythms. Another limitation is that to achieve high accuracy on some complex pieces, our system still requires some human effort to choose the optimal parameter setting, at least on the page level. The uniform parameter setting is too tolerant on very complex interpretations, thus leads to errors on easier measures.

Rhythm interpretation in OMR is worth exploring in many future directions. Currently, we are only using the measure level music knowledge to understand rhythm. But there is more structural information (such as repetition and analogy) on the scale of

neighboring measures, pages, or movements. How to model these structures would be an interesting problem. On the other hand, how to more effectively use human inputs is another direction worth considering.

Reference:

[1] D. Bainbridge and T. Bell, "The challenge of optical music recognition," *Computers and the Humanities*, vol. 35, pp. 95–121, 2001.

[2] M. Droettboom, I. Fujinaga, and K. Macmillan, "Optical music interpretation," *Structural, Syntactic, and Statistical Pattern Recognition*, pp. 378–387, 2002.

[3] G. Read, *Music notation: a manual of modern practice*. Taplinger Publishing, 1969.

[4] "SmartScore." [Online]. Available: http://www.musitek.com/.

[5] D. Pruslin, "Automatic recognition of sheet music (PhD Thesis)."

[6] D. Prerau, "Computer pattern recognition of standard engraved music notation (PhD Thesis)," 1970.

[7] I. Fujinaga, "Adaptive optical music recognition (PhD Thesis)," 1996.

[8] I. Fujinaga, "Optical music recognition using projections (Master Thesis)," 1988.

[9] L. S. Johansen, "Optical music recognition (Master Thesis)," 2012.

[10] A. Rebelo, "Robust optical recognition of handwritten musical scores based on domain knowledge," *Pattern Recognition (ICPR)*, 2012.

[11] D. Bainbridge, "Extensible optical music recognition (PhD Thesis)," 1997.

[12] D. Blostein and H. S. Baird, "A critical survey of music image analysis," *Structured Document Image Analysis*, pp. 405–434, 1992.

[13] A. Rebelo, I. Fujinaga, F. Paszkiewicz, A. R. S. Marcal, C. Guedes, and J. S. Cardoso, "Optical music recognition: state-of-the-art and open issues," *International Journal of Multimedia Information Retrieval*, vol. 1, no. 3, pp. 173–190, 2012.

[14] J. Novotny and J. Pokorny, "Introduction to optical music recognition: Overview and practical challenges," *CEUR Workshop Proceedings*, vol. 1343, pp. 65–76, 2015.

[15] A. Rebelo, G. Capela, and J. S. Cardoso, "Optical recognition of music symbols," *International Journal on Document Analysis and Recognition*, vol. 13, no. 1, pp. 19–31, 2010.

[16] I. Fujinaga, *Visual perception of music notation: Online and offline recognition*. Idea Group Inc, 2003.

[17] K. Ng, *Interactive multimedia music technologies*. Hershey, 2008.

[18] H. Bunke, K. Yamamoto, and H. S. Baird, *Structured document image analysis*. 1992.

[19] "SharpEye." [Online]. Available: http://www.visiv.co.uk/.

[20] "Capella-scan." [Online]. Available: https://www.capella-software.com/us/.

[21] "PhotoScore." [Online]. Available: http://www.neuratron.com/.

[22] C. Raphael and R. Jin, "Optical music recognition on the international music score library project," *Proceedings of SPIE - The International Society for Optical Engineering*, vol. 9021, pp. 1–12, 2014.

[23] K. MacMillan, M. Droettboom, and I. Fujinaga, "Gamera: optical music recognition in a new shell," in *Proceedings of the international computer music conference*, 2002.

[24] F. Rossant and I. Bloch, "Robust and adaptive OMR system including fuzzy modeling, fusion of musical rules, and possible error detection," *Eurasip Journal on Advances in Signal Processing*, vol. 2007, 2007.

[25] M. Szwoch, "Guido: A musical score recognition system," *Proceedings of the International Conference on Document Analysis and Recognition, ICDAR*, vol. 2, no. 3, pp. 809–813, 2007.

[26] J. A. Burgoyne, L. Pugin, G. Eustace, and I. Fujinaga, "A comparative survey of image binarisation algorithms for optical recognition on degraded musical sources," *ACM SIGSOFT Software Engineering Notes*, vol. 24, no. 1985, pp. 1994–1997, 2008.

[27] T. Pinto, "Music score binarization based on content knowledge," pp. 1–8, 2010.

[28] J. S. Cardoso and A. Rebelo, "Robust staffline thickness and distance estimation in binary and gray-level music scores," *Proceedings - International Conference on Pattern Recognition*, pp. 1856–1859, 2010.

[29] J. S. Cardoso, A. Capela, A. Rebelo, and C. Guedes, "A connected path approach for staff detection on a music score," *Proceedings - International Conference on Image Processing, ICIP*, pp. 1005–1008, 2008.

[30] J. Cardoso, A. Capela, A. Rebelo, C. Guedes, and J. Costa, "Staff detection with stable paths," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 6, pp. 1134–1139, 2009.

[31] H. Miyao, "Stave extraction for printed music scores," *Intelligent Data Engineering and Automated Learning - IDEAL*, pp. 562–568, 2002.

[32] F. Alirezazadeh and M. R. Ahmadzadeh, “Effective staff line detection, restoration and removal approach for different quality of scanned handwritten music sheets,” *Journal of Advanced Computer Science & Technology*, vol. 3, no. 2, pp. 136–142, 2014.

[33] B. Su, S. Lu, U. Pal, and C. L. Tan, “An effective staff detection and removal technique for musical documents,” *Proceedings - 10th IAPR International Workshop on Document Analysis Systems*, pp. 160–164, 2012.

[34] C. Dalitz, M. Droettboom, B. Pranzas, and I. Fujinaga, “A comparative study of staff removal algorithms,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, no. 5, pp. 753–766, 2008.

[35] F. Toyama, K. Shoji, and J. Miyamichi, “Symbol recognition of printed piano scores with touching symbols,” *International Conference on Pattern Recognition*, vol. 2, pp. 480–483, 2006.

[36] I. Fujinaga, “Exemplar-based learning in adaptive optical music recognition system,” *Proceedings of the international computer music conference*, pp. 55–56, 1996.

[37] P. Bellini, I. Bruno, and P. Nesi, “Optical music sheet segmentation,” *Proceedings - 1st International Conference on WEB Delivering of Music, WEDELMUSIC 2001*, no. December, pp. 183–190, 2001.

[38] K. T. Reed and J. R. Parker, “Automatic computer recognition of printed music,” *Proceedings - International Conference on Pattern Recognition*, vol. 3, pp. 803–807, 1996.

[39] A. A. Mehta and M. S. Bhatt, “Optical music notes recognition for printed piano music score sheet,” *2015 International Conference on Computer Communication and Informatics*, pp. 8–13, 2015.

[40] C. Wen, J. Zhang, A. Rebelo, and F. Cheng, “A directed acyclic graph-large margin distribution machine model for music symbol classification,” *plos one*, vol. 11, no. 3, 2016.

[41] B. Coüasnon and J. Camillerapp, “Using grammars to segment and recognize music scores,” 1994.

[42] B. Coüasnon, “Using a grammar for a reliable full score recognition system.” , 1995.

[43] L. Pugin, “Optical music recognition of early typographic prints using Hidden Markov Models,” pp. 3–6, 2006.

[44] G. E. Kopec, P. A. Chou, and D. A. Maltz, “Markov source model for printed music

decoding," vol. 1, pp. 1–11, 1996.

[45] C. Raphael and J. Wang, "New approaches to optical musicrecognition," *12th International Society for Music Information Retrieval Conference*, pp. 305–310, 2011.

[46] M. Church and M. Cuthbert, "Improving rhythmic transcriptions via probability models applied post-OMR," *15th International Society for Music Information Retrieval Conference*, pp. 643–648, 2014.

[47] D. Byrd, "Prospects for improving OMR with multiple recognizers," *7th International Society for Music Information Retrieval Conference*, pp. 41–46, 2006.

[48] E. Bugge, L. Juncher, B. Soborg, M. Jakob, and G. Simonsen, "Using sequence alignment and voting to improve optical music recognition from multiple recognizers," *International Society for Music Information Retrieval Conference*, pp. 405–410, 2011.

[49] V. Padilla, A. Mclean, A. Marsden, and K. Ng, "Improving optical music recognition by combining outputs from multiple sources," *16th International Society for Music Information Retrieval Conference*, pp. 517–523, 2015.

[50] R. Jin and C. Raphael, "Interpreting rhythm in optical music recognition," *13th International Society for Music Information Retrieval Conference*, no. Ismir, pp. 151–156, 2012.

[51] R. Jin and C. Raphael, "Graph-based rhythm interpretation," *16th International Society for Music Information Retrieval Conference*, 2015.

[52] D. Byrd and E. March, "Towards a standard testbed for optical music recognition : definitions , metrics , and page images," *Journal of New Music Research*, no. 44, pp. 169–195, 2015.

[53] P. Bellini, I. Bruno, P. Nesi, and V. S. Marta, "Assessing optical music recognition tools evaluation methodologies for OMR application."

[54] M. Droettboom and S. S. West, "Symbol-level groundtruthing environment for OMR," *Gamut*.

[55] M. Szwoch, "Using musicXML to evaluate accuracy of OMR systems," *Lecture Notes in Computer Science*, vol. 5223 LNAI, pp. 419–422, 2008.

[56] J. Hook, "How to perform impossible rhythms," *Music Theory Online*, vol. 17, no. 4, pp. 1–14, 2011.

[57] D. Byrd, "Gallery of interesting music notation." [Online]. Available: http://homes.soic.indiana.edu/donbyrd/InterestingMusicNotation.html.

[58] D. Byrd, “Music notation by computer,” (PhD Thesis),1984.

[59] D. Byrd, “Extremes of conventional music notation,” 2016. [Online]. Available: http://homes.soic.indiana.edu/donbyrd/CMNExtremes.htm.

[60] D. Byrd, “More counterexamples in conventional music notation.” [Online]. Available: http://homes.soic.indiana.edu/donbyrd/MoreCMNCounterexamples.htm.

[61] R. Bellman, “The theory of dynamic programming,” *Bulletin of the American Mathematical Society*, vol. 60, no. 6. pp. 503–515, 1954.

[62] T. Cormen, C. Leiserson, and R. Rivest, *Introductin to algorithms*. 1989.

[63] P. Felzenszwalb and R. Zabih, “Dynamic programming and graph algorithms in computer vision,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 4, pp. 1–52, 2011.

[64] L. Rabiner, “A tutorial on Hidden Markov Models and selected applications in speech recognition,” *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, 1989.

[65] C. Raphael, “Automatic segmentation of acoustic musical signals using Hidden Markov Models,” vol. 21, no. 4, pp. 360–370, 1999.

[66] D. Ellis and G. Poliner, “Identifying ‘cover songs’ with chroma features and dynamic programming beat tracking,” *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, vol. 4, 2007.

[67] C. Raphael, “A hybrid graphical model for aligning polyphonic audio with musical scores,” *Machine learning*, vol. 65, no. 2–3, pp. 389–409, 2006.

[68] D. Forney, “The viterbi algorithm,” *Proceedings of the IEEE*, vol. 61, no. 3, pp. 268–278, 1973.

[69] F. Kschischang, “Factor graphs and the sum-product algorithm,” *Journal of Chemical Information and Modeling*, vol. 53, no. 9, pp. 1689–1699, 2013.

[70] S. Lauritzen and D. Spiegelhater, “Local computations with probabilities on graphical structures and their applications to expert systems,” *Journal of the Royal Statistical Society*, vol. 25, no. 2, pp. 264–296, 2016.

[71] G. Shafer, “Probability judgment in artificial intelligence and expert systems,” *Statistical Science*, vol. 2, no. 1, pp. 3–16, 1986.

[72] G. Shafer and P. Shenoy, “Probability propogation,” *Annals of Mathematics and Artifical Intelligence*, vol. 2, pp. 327–253, 1990.

[73] A. Dawid, “Applications of a general propagation algorithm for probabilistic

expert systems," *Statistics and Computing*, vol. 2, no. 1, pp. 25–36, 1992.

[74] S. Andersen and K. Olesen, "HUGIN *-- a shell for building Bayesian belief universes for expert systems," *International Joint Conferences on Artificial Intelligence*, 1989.

[75] H. L. Bodlaender, "Dynamic programming on graph with bounded treewidth," *Proc. 15th International Colloquium on Automata, Languages and Programming*, 1988.

[76] M. Bern, E. Lawler, and A. Wong, "Linear-time computation of optimal subgraphs of decomposable graphs," *Journal of Algorithms*, vol. 8, no. 2, pp. 216–235, 1987.

[77] S. Arnborg and A. Proskurowski, "Linear time algorithms for NP-hard problems restricted to partial k-trees," *Discrete Applied Mathematics*, vol. 23, no. 1, pp. 11–24, 1989.

[78] L. Chen, E. Stolterman, and C. Raphael, "Optical music recognition with human-labeled constraints," *ACM SIGCHI Workshop on Human-Centered Machine Learning*, pp. 0–5, 2016.

[79] L. Chen and C. Raphael, "Human interactive optical music recognition," *International Society for Music Information Retrieval Conference*, 2016.

[80] "International Music Score Library Project." [Online]. Available: http://imslp.org/.

[81] "MusicXML." [Online]. Available: http://www.musicxml.com/tutorial/the-midi-compatible-part/multi-part-music/.

Curriculum Vitae

Rong Jin

Music Informatics

School of Informatics, Computer Science and Engeneering

Indiana University Bloomington

Education

Indiana University (Bloomington, IN) 2010 - 2017

Ph.D. in School of Informatics and Computing

Tsinghua University (Beijing, China) 2006 - 2010

B.S. in School of Information Science, Automation Department

Professional Experience

Research Assistant (Indiana University Bloomington) 2010 – 2017

Teaching Assistant (Indiana University Bloomington) 2010 – 2017

Data Analyst intern (Megaputer Intelligence Inc., IN) 2013 summer

Publication

1. L Chen, Y Jiang, R Jin, etc. Optical Music Recognition with Human-Labeled Constraints. *Human-Centered Machine learning (HCML) workshop*, 2016

2. L Chen, R Jin, etc. A hybrid HMM-RNN model for OMR. *The International Society of Music Information Retrieval conference, 2016*.

3. R Jin, C Raphael. Graph-based Rhythm Interpretation. *The International Society of Music Information Retrieval conference, 2015.*

4. L Chen, R Jin, C Raphael. Renotation from Optical Music Recognition. *Mathematics and Computation in Music, 2015.*

5. R Jin, C Raphael. *Optical music recognition on the international music score library project. IS&T/SPIE Electronic Imaging, 2013.*

6. R Jin, C Raphael. Interpreting Rhythm in Optical Music Recognition. *The International Society of Music Information Retrieval conference, 2012.*

Skills

Professional training in machine learning, computer vision, statistics.

Extensive experience in probabilistic graphical models, deep learning, NLP, audio processing.

Extensive hands-on programming experience in C, C++, Objective C, Python.

Proficient in R, Matlab, Bash, Java, MySOL, Javascript.

Efficient communication skills.