A Geometric Approach to Pattern Matching in Polyphonic Music

by

Luke Andrew Tanur

A thesis
presented to the University of Waterloo
in fulfilment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2005

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

The music pattern matching problem involves finding matches of a small fragment of music called the "pattern" into a larger body of music called the "score". We represent music as a series of horizontal line segments in the plane, and reformulate the problem as finding the best translation of a small set of horizontal line segments into a larger set of horizontal line segments. We present an efficient algorithm that can handle general weight models that measure the musical quality of a match of the pattern into the score, allowing for approximate pattern matching.

We give an algorithm with running time $O(nm(d + \log m))$, where n is the size of the score, m is the size of the pattern, and d is the size of the discrete set of musical pitches used. Our algorithm compares favourably to previous approaches to the music pattern matching problem. We also demonstrate that this geometric formulation of the music pattern matching problem is unlikely to have a significantly faster algorithm since it is at least as hard as 3SUM, a basic problem that is conjectured to have no subquadratic algorithm. Lastly, we present experiments to show how our algorithm can find musically sensible variations of a theme, as well as polyphonic musical patterns in a polyphonic score.

Acknowledgments

I would first like to extend my utmost gratitude to my supervisor, Anna Lubiw. It goes without saying that I could not have accomplished any of this work without her guidance, patience, and encouragement. I am further indebted to her for supplying me with research assistantship funding from her research grant.

I would also like to thank my readers, Dan Brown and Charlie Clarke, for supplying me with valuable feedback, and also for agreeing to read this thesis on relatively short notice.

Credit for the idea of how to modify our algorithm to find the best k matches must go to Ian Munro—thank you.

Thanks must also go to those who have provided me with a great deal of support over the past eight months, when I have needed it most. My parents, Peter and Anna, as well as my sisters, Adrienne and Cheryl, are at the top of that list. To all of my close friends who have always been there for me—Alex, Jen, Magda, Phil, and Raymond—thank you. I could not have made it this far without all of you.

Finally, I would like to thank the School of Computer Science for providing me with funding through teaching assistantships during my stay here.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

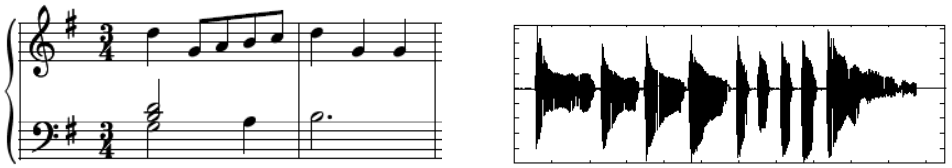
Introduction

Music information retrieval is a young, rapidly developing area of research. While popular systems such as Internet search engines and other applications exist to search for music based on a textual query, bibliographic data (such as a song title or the name of the composer) is only one defining characteristic of a piece of music. The general goal of music information retrieval is to extract information of some kind from music stored in a digital format. In particular, how does one search for a piece of music if the title, artist, and composer are all unknown, but a fragment of the melody of the piece is known? This type of problem is a main focus of current research in music information retrieval.

Compared to the well-established field of text information retrieval, which is firmly rooted in both computer science and library science, research contributions to music information retrieval come from many disparate fields. In addition to computer scientists, researchers from the fields of music theory, library science, digital signal processing, cognitive science, and law have made important contributions. These different perspectives lead to many different research areas within music information retrieval, and the interdisciplinary nature of the field has many benefits and drawbacks.

In text information retrieval, a typical task would involve a search query that consists of one or more words, or phrases. The information retrieved may be in the form of web pages, articles, or just titles of larger bodies of text. It is a complex problem to interpret the search query, and return the most relevant results to the user even when the only thing to be considered is text. Music, on the other hand, is significantly more complicated because there are many different ways that music can be represented. For each representation, different musical features may be either present or absent, and if present the difficulty of extracting that feature from a particular representation may vary. See Figure 1.1. Moreover, musical features are often not independent—for example, the musical harmony that corresponds to a certain chord also depends on the pitch and timing of appropriate notes. Because music is so multi-faceted, it is necessary to allow

for richer ways of extracting musical information beyond the standard one-dimensional textual query, which provides us with only an imprecise way to search for music.

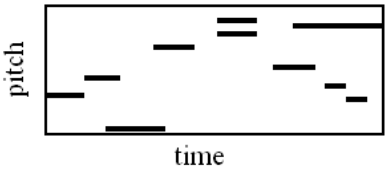


(a)

(b)

(60, 2), (64, 2), (65, 4),
(70, 1), (69, 1), (67, 2)

(c)



(d)

Figure 1.1: Different musical representations: (a) J. S. Bach, Menuet, BWV Anh. 114, bars 1–2, in common musical notation (CMN) for piano; (b) a time-amplitude representation of a sung melody, adapted from Shifrin et al. [47]; (c) a string of ordered (*pitch*, *duration*) pairs; (d) a "piano roll" representation using line segments.

We study one of the problems at the core of music information retrieval, that of "music pattern matching". In the context of a symbolic representation of music, the music pattern matching problem is to find the "best" match of a small fragment of music, called the *pattern*, into a large body of music often representing a single musical work, called the *score*. Our approach focuses on the musical features of pitch and time, representing a musical work as a series of horizontal line segments in the plane. Each line segment corresponds to a single musical note, with its vertical coordinate corresponding to the pitch of the note, and the horizontal coordinates of its endpoints corresponding to the times at which the note starts and ends.

We present an efficient algorithm that solves the music pattern matching problem when both the pattern and the score are polyphonic. Furthermore, we provide a flexible way for the user to specify how a "good" match should be defined, using fairly general weight functions for different types of approximate pattern matching. This will allow matches of the pattern into the score that are meaningful, in a musical sense. This approach can be easily extended to discover the k best matches of a pattern into a score.

1.1 Thesis Outline

This thesis is organized as follows. We first present an overview of the field of music information retrieval in Chapter 2, as well as some simple musical concepts for background information. In Chapter 3, we discuss previous work done in the field of music information retrieval and relate these concepts to our work. Chapter 4 presents the theoretical aspects of our music pattern matching algorithm, including arguments proving correctness and time complexity. In Chapter 5, we present reasons why finding a more efficient algorithm to solve the music pattern matching problem in the geometric setting is difficult and discuss the methodology and results of our experiments in Chapter 6. Lastly, we present our conclusions and potential avenues for future work in Chapter 7.

Chapter 2

Background

It is only in the last decade that the field of music information retrieval has blossomed, although research in the field dates as far back as the 1960's (see Kassler [25]). While research in the traditional field of text information retrieval steadily progressed due to the ease of storing and searching textual data, it was not until digital storage of music became common in the mid-1990's that a surge of interest in music information retrieval took place. As the costs of data storage and bandwidth decreased, larger collections of music in digital form became more prevalent, and the need for different ways of extracting information from these collections grew. The Music Information Retrieval Annotated Bibliography [12] provides a rough idea of the frequency of contributions over time.

Because music information retrieval overlaps with so many different disciplines, initial research in the area did not have a venue that solely focused on music information retrieval. Results that deal primarily with music theory would be published in that domain, a result extending speech recognition to applications in music information retrieval might be published in an artificial intelligence journal, and extensions of traditional information retrieval techniques to music would be presented at conferences on digital libraries. The problem of bringing researchers in music information retrieval together from all fields started to be addressed with dedicated workshops and panels at conferences on computer music, information retrieval, and digital libraries. Starting in 2000, an annual conference dedicated to music information retrieval was formed. Paper submissions and attendance at each International Conference on Music Information Retrieval (ISMIR) have increased in every year [2].

To gain a better understanding of the current research goals in music information retrieval, it is necessary to identify each research discipline that makes significant contributions to the field, and to explain the nature of these contributions. We use the categories given in Futrelle and Downie's examination of interdisciplinary issues [18].

We then discuss limitations of existing music information retrieval systems. Before specifying the nature of the music information retrieval problem that we address, we must

provide background on common musical features, and also distinguish between audio and symbolic representations of music.

2.1 Relevant Disciplines

We can divide the disciplines involved in music information retrieval into six groups. First, we have the computer scientists and those from the information retrieval discipline, focusing mainly on software and algorithms. Another group includes those who specialize in audio engineering and digital signal processing, extending previous work in speech recognition to deal with audio representations of music such as Moving Picture Experts Group-1/2 Audio Layer 3 (MP3) files. Music theorists and musicologists wish to use music information retrieval systems to aid in the analysis of music. Those in library science focus on user interfaces, as well as user studies to figure out what potential users want from a music information retrieval system. Cognitive scientists, psychologists, and philosophers highlight an important viewpoint by studying how humans perceive music. Finally, those in the field of law deal with the issue of intellectual property rights relating to digital music, which has been prevalent in the media in recent years.

Techniques from *computer science* are present to some extent in most results from music information retrieval research. However, there is a particular emphasis on traditional information retrieval, rooted in the early days of text retrieval systems. A basic text retrieval system would take a word as input, and search for documents that contain that word. The analogous task in a musical setting would be to take some symbolic representation of a musical fragment as input, and find musical works that contain that fragment. We will examine some of the different symbolic representations used in Chapter 3.

Audio representations of music include recordings, live performances, and raw audio files. Related work dates back to the early days of *digital signal processing*, as well as the intervening decades of research on speech recognition and audio compression techniques. The primary focus of this discipline is to extract musical features from these audio representations for analysis and classification.

A purely academic application of music information retrieval research is to aid those studying *music theory* in the study and analysis of music. Selfridge-Field [44] discusses the advanced needs of music theorists and musicologists, compared to the needs of a casual user of a music information retrieval system. Applying computational techniques to a traditionally qualitative task such as analyzing the style of a particular musical work raises many challenges, and input from those with a music theory background is necessary for a useful system.

Researchers in the discipline of *library science* are concerned with understanding user needs in the context of music information retrieval. Libraries around the world have rapidly growing collections of music, and figuring out the best way to deal with such

collections opens up many areas of research. Issues such as determining how to index material in these collections, as well as how to best represent the music stored in these collections are important elements of any music information retrieval system. Also of interest in this discipline is the adaptation of existing bibliographic systems to interact with music collections.

Those studying *music cognition* make up a small, but important, group within the music information retrieval community. Drawing from disciplines such as cognitive science, psychology, and philosophy, researchers in this area tackle the issues of how humans perceive and remember music. It is unreasonable for a human to perfectly reproduce certain aspects of a musical work, and so more work needs to be done to determine how music cognition may affect the quality of a human user's query to a music information retrieval system.

Legal issues are closely tied to the future of research in music information retrieval. Recent developments such as the legal battles over file sharing systems such as Napster, the emerging growth of online music stores, as well as new means of music piracy serve to emphasize the uncertain legal environment inherent to the distribution of digital music. Although copyright law is not a technical issue that can be addressed by the music information retrieval community, there are sizable commercial implications to however these issues are resolved. These legal issues also directly affect how music information retrieval projects are funded, as well as how much music is accessible by researchers in the field.

The interdisciplinary nature of music information retrieval is both a blessing and a curse. For example, many research projects necessarily involve members of multiple disciplines, which usually results in a finding that is less limited by the constraints of one particular discipline, and hence has broader appeal to a typical user. On the other hand, terminology and concepts that are commonly used by computer scientists can be meaningless to music theorists, and vice versa. Therefore in order to gain a better understanding of a particular music information retrieval project, it may be necessary to spend a non-trivial amount of time learning some of the basics of other disciplines. Also, costly duplication of effort may take place if a research group in one discipline is not aware of a relevant advance in another discipline.

2.2 Current Research Areas

There are many different research areas and goals in music information retrieval. We will give a brief overview of each of these areas, and then describe several potential applications that can arise from research in this field.

Music representation is an important research area dealing with methods for representing digital music, as well as methods used to extract musical features from such representations. Sparse representations are not demanding in terms of data storage space,

but can only encode limited information about few musical features. Full-fledged representations may be able to encode a great deal of information about many musical characteristics, but storing a large musical collection using such representations may be extremely impractical. An appropriate trade-off between space used and features gained must be achieved, and the question of which musical features should be represented is relevant to many applications. A typical application can involve user-based experiments to determine whether a particular representation is suitable for a user query (see Uitdenbogerd and Yap [50]).

Indexing and *retrieval* are two closely related concepts. From previous work in the databases field, different indexing methods can be applied to musical collections in order to retrieve musical data efficiently. Determining the nature of user queries, as well as evaluating the performance of user queries are two main retrieval tasks. From these two perspectives, the performance of a system can be improved by both improving the indexing techniques applied to the musical database, as well as designing the system to perform well on a certain class of user queries. One application of indexing can be found in Downie and Nelson's examination of n-grams [13]. For examples of retrieval, see Lemström et al. [31] as well as Shifrin and Birmingham [46].

User interface design focuses on making systems more accessible to users. Users should find music information retrieval systems easy to use, and it should also be easy for users to successfully find the information that they desire. It should be easy for a user to input a user query into the system that corresponds to the user's needs, and some work in this area focuses on interfaces that use speech recognition (as in Goto et al. [20]). The goal of *recommendation* systems is to automatically recommend similar music that the user may enjoy, based on a user profile incorporating existing musical preferences (see Logan [32]). *User studies* are not tied to a specific system, and instead determine what capabilities particular user communities may want from a music information retrieval system—for example, see Lee and Downie [28].

The main research goals concerning audio representations of music revolve around *compression* and *feature detection*. Audio compression deals with more efficient encodings of music, which can lead to reductions in the size of musical collections. An audio fingerprint is one example of a compact musical representation (as in Doets and Lagendijk [9]). Feature detection examines the question of how to extract meaningful musical features from an audio representation, as well as how such techniques can be incorporated into existing systems (see Pauws [42]).

Classification of music allows musical collections to be divided into sub-collections of "similar" music—see McKinney and Breebaart [36]. The notion of similarity can be difficult to pin down, but methods (such as machine-learning) can be applied to a body of music, and then *musical analysis* could be applied to ensure that grouping music into those sub-collections makes musical sense. The area of musical analysis perhaps most closely ties into the notion of musical similarity, and incorporating this musicological

perspective into a system remains an important goal. Other tasks in musical analysis include musical instrument recognition from audio (as in Eggink and Brown [15]), as well as harmonic analysis (see Sheh and Ellis [45]).

Somewhat related to the area of representation are the concepts of *summarization* and *metadata*. Research in summarization attempts to automatically summarize a musical work, and also addresses the question of what musical elements should be in the generated summary (as in Cooper and Foote [8]). Metadata is defined to be descriptive information associated with a musical work. Common examples of metadata include the performer, title, and composer of a work, and existing music file sharing systems primarily use metadata to find music. How metadata should be represented and used in a music information retrieval system are two important issues in this area—see Olson and Downie [40].

The legal issues surrounding the area of *intellectual property rights* affect all groups in the music information retrieval community—for example, see Chiariglione [7]. Questions of who exactly owns musical material necessarily limit the amount of musical material that researchers have access to, and also make it difficult to build extremely large collections of music for experimental purposes. Key issues in this area involve determining common goals for owners of musical material as well as maintainers of digital libraries, and figuring out what incentives musical content providers need to make music more accessible for research purposes.

Perception deals with the ways that humans perceive and remember music, as in Lartillot [27]. In particular, a particular person may perceive two pieces of music as "similar", even though there is no obvious similarity when viewed from a classical musical analysis viewpoint. Determining the musical features that have a non-trivial impact on how we perceive music provides valuable insight into how music information retrieval systems should be constructed. Lastly, *epistemology* addresses the more basic questions concerning what music actually is—see Smiraglia [48]. Relationships between different representations of a musical work, as well as the relationship between improvisation and composition are also studied in this area.

2.2.1 Potential Applications

Many potential applications can arise from advances in music information retrieval. With the emergence of services that sell music in various audio formats online like the iTunes Music Store, and the growing popularity of portable MP3 players, the demand for digital music is continuing to increase [24]. Sales of compact discs continue to be strong, with a 2.8% increase in CD volume growth in the US from 2003 to 2004 (see [39]). The vast commercial demand for music provides many opportunities for research in music information retrieval, as systems that are better able to find "relevant" pieces of music for the user will provide a significant competitive edge. Moreover, existing underutilized

music collections in libraries around the world can be made accessible to a wider audience with the development of a robust music information retrieval system [10].

The importance of being able to search for music in different ways can be demonstrated by a few examples. The currently available metadata query using bibliographic data is unlikely to become obsolete, but advances in music information retrieval can better allow a consumer to find music "similar" to that performed by a certain artist, with the consumer being able to define the aspects of that similarity. For example, a music information retrieval system may receive a particular artist as user input, scan through the artist's works, and drawing from a larger musical database, outputs works by different artists that have similar chord progressions to the works of the original artist.

The "query by humming" approach involves a user input of a melody; that is, the user hums or sings a tune. A music information retrieval system could record the user input and find musical works in a database that have similar melodies, which can be useful if the user wishes to know more about musical works that have some degree of similarity with the melody. If the user wishes to add further information to the query, such as a list of composers for example, the measure of similarity used to output musical works can be refined further.

2.3 Limitations of Current Music Information Retrieval Systems

There are many different issues that current music information retrieval systems have trouble dealing with [6]. In the query by humming example, the user will probably be out of tune, or will not remember the actual tune precisely. If this is the case, then the pitches recorded by the system may differ from the actual melody that the user intended to hum. In addition to user error, the pitch tracker itself may not be able to transcribe the pitches in the audio signal correctly. Therefore error correction techniques need to be considered to predict what the actual user melody is.

Furthermore, the vast majority of music is *polyphonic*, with multiple "voices" interacting with each other at the same time. See Figure 2.1. A search query consisting of a monophonic melody may not yield relevant results when examining a polyphonic piece of music. For example, a naive pattern matching algorithm might yield a meaningless approximate match of the query within a single voice of a polyphonic work, when a much better match would span multiple voices in the same work. It is for this reason that music information retrieval techniques analogous to string pattern matching algorithms used in text information retrieval do not generally work well for polyphonic music—musical data over multiple voices cannot be easily represented as a one-dimensional string. Some existing approaches to dealing with polyphony in a music information retrieval task are discussed in Section 3.2.

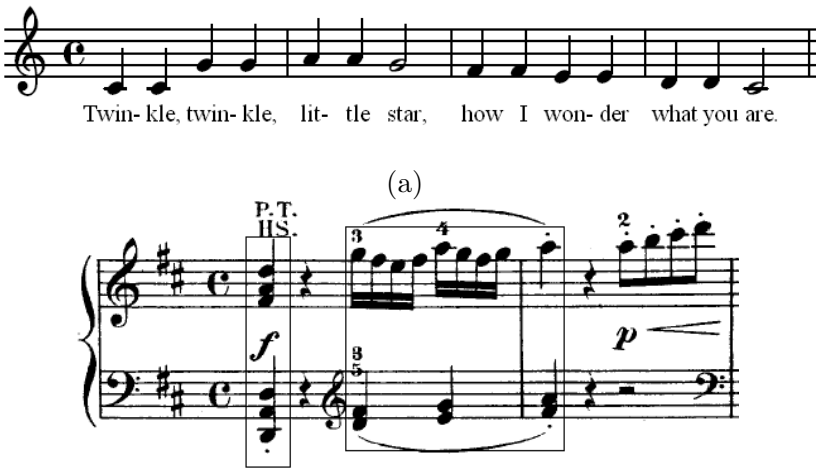


Figure 2.1: (a) "Twinkle, Twinkle, Little Star", bars 1–4, an example of monophonic music with only one note sounding at any point in time; (b) W. A. Mozart, Piano Sonata in D major K311, 1st movement, bars 1–2, an example of polyphony. Boxes indicate where multiple notes are being played at the same time.

Often recognizable themes found in a musical work are prone to variation over the course of that work. While at the beginning of a piece of music, a particular melody may appear as an exact match to a user query, restatements of this theme later on in the piece could have slight changes, ranging from simple alterations of rhythm, to complex additions of ornamentations. This underscores the need for *approximate matching*; we would like to be able to recognize variations of a theme, in some way, as only finding exact matches to a query is too limiting.

One of the main issues at the heart of music information retrieval is how to specify the relevance of an approximate match—in particular, what combination of musical features allow us to determine whether two musical fragments are similar. For example, whether a musical pattern is a variation of a particular theme depends on this interplay of multiple interdependent musical features. While a trained music theorist is able to make such a distinction, music information retrieval systems that can make accurate judgements of similarity or dissimilarity have not yet been realized. Also, certain user communities may have a different idea about how similarity is defined, when compared to the common musicological point of view. For example, the music of certain cultures do not follow the same musical conventions as standard Western music, and so the corresponding notions of musical similarity may also be different.

The above limitations represent a few symptoms of three more fundamental problems with research in the field, as identified by Futrelle and Downie [18]. The first problem is

that it is difficult to evaluate, let alone compare, different music information retrieval techniques. Different disciplines have different evaluation metrics, and no standard evaluation methods exist for particular classes of music information retrieval problems. Evaluation methods are rarely explained properly, and there has been relatively little work done on large musical collections. The development of common musical collections for multiple techniques to be tested on is one method of attacking this problem, and attempts to devise standardized testbeds and evaluation metrics such as the ones available to the text information retrieval field are being explored (see [11, 23]).

The second problem is that few attempts to thoroughly assess user needs have been made, compared to the vast number of projects that simply assume that a particular approach (for example, query by humming) is most useful. Part of the difficulty of assessing real user needs is the lack of usable large-scale systems. As testbed projects continue to evolve, user studies will start to yield more meaningful information; however, to address this problem, greater emphasis on user interfaces and user needs is required.

Lastly, the vast majority of research in music information retrieval is focused on standard Western music. This emphasis is not too surprising, as the majority of researchers in the field are based in Europe and North America. However, there are many non-Western musical forms such as Indian ragas, African tribal music, and Japanese koto music, that cannot be adequately handled by most common music information retrieval systems. This is because there are certain properties inherent to Western music, and systems generally assume that these properties hold true for all musical works in the collection. Addressing this problem will require researchers to re-evaluate assumptions about music, and more closely examine the cultural aspects of real user communities.

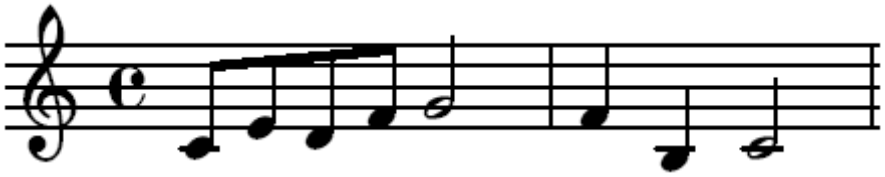
2.4 Musical Features

Content-based music information retrieval allows the user query to incorporate musical features other than textual metadata. A content-based query may be comprised of any combination of musical features, including note pitches, temporal or rhythmic information, chord symbols for harmony, what types of musical instruments are present, song lyrics for vocal works, and bibliographic data, among others.

A major issue in music information retrieval is how to weigh all of these different musical features appropriately. It is meaningless to address each of them individually, because of the inherent interdependence of many of these features in most musical works. Similarly, it seems infeasible to work towards a unique standard for a content-based query that would weigh the importance of each musical feature in a certain way, because there are many different types of music, each of which places different emphases on various features. It is also important to consider the difficulty of extracting certain musical features from a user query. We give a more in-depth explanation of notable musical features and associated terminology below.

The *pitch* of a musical note corresponds to the frequency of that note. For example, in modern music, the note A above middle C on a piano keyboard is the sound with frequency 440 Hz. There are many different ways to represent the pitch of a note, the most common involving the vertical position on the musical staff. In MIDI files, the pitch of a note is specified by a number between 0 and 127. These numbers correspond to musical semitones, where a semitone is the difference in pitch between two consecutive notes on a piano keyboard, roughly a 6% difference in frequency. Therefore middle C corresponds to MIDI pitch 60, and the A above middle C corresponds to MIDI pitch 69.

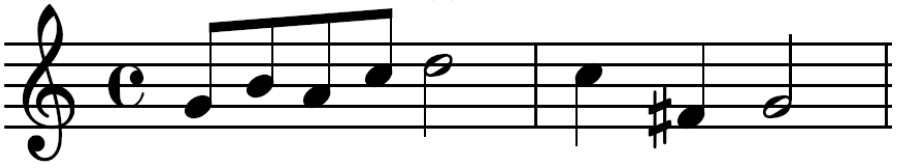
An *interval* is the difference between different pitches. We use MIDI pitches to specify intervals in this thesis, so the interval between middle C and the A above middle C is nine semitones. In Western music, it is often the case that a musical work can be divided into several significant parts, each part based on a particular *key*. For example, one section of music could be in the key of C major, while the next section could be in the key of G major. Since the interval between C and G is seven semitones, we can *transpose* a musical fragment from C major to G major by simply shifting the pitch of each note up by seven semitones. We consider the two musical fragments in Figure 2.2 to be identical under transposition, and for the purposes of this thesis, such musical fragments are musically equivalent.



MIDI pitch: 60 64 62 65 67 65 59 60

Interval (in semitones): +4 −2 +3 +2 −2 −6 +1

(a)



MIDI pitch: 67 71 69 72 74 72 66 67

Interval (in semitones): +4 −2 +3 +2 −2 −6 +1

(b)

Figure 2.2: (a) a musical fragment in C major, with associated MIDI pitches and intervals indicated; (b) the same musical fragment transposed up seven semitones, to G major.

Time affects many aspects of music—the tempo, or speed at which music is played, durations of notes and rests, and the meter of the musical work, which determines the basic time units that the musical work can be divided into. For example, a time signature

of 6/8 assigns six 8^{th} notes to every *bar*, so that each bar in the musical work contains six beats, each the length of one 8^{th} note. See Figure 2.3. The meter also implicitly determines which beats should be emphasized, or accented. All of these temporal factors interact to produce the *rhythmical* features of music.

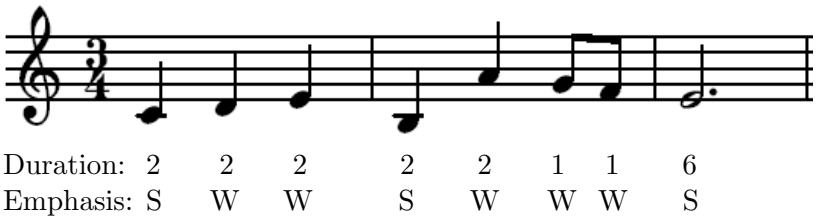


Figure 2.3: Rhythmical features of music. This musical fragment consists of three bars in 3/4 time, and units of duration correspond to 8^{th} notes.

A monophonic sequence of musical notes in time can form a *melody*—thus a melody is produced from the interaction of pitches and time. The best example of a melody would be simply singing a tune. When several musical notes with different pitches overlap in time (polyphony), *harmony* is created. Often simultaneous notes form a *chord*, which can be thought of as a stand-alone harmonic unit. Musical conventions exist to define properties of certain chords, and more importantly conventions also exist to guide movement from one chord to the next in time. A sequence of chords is called a chord progression; see Figure 2.4.

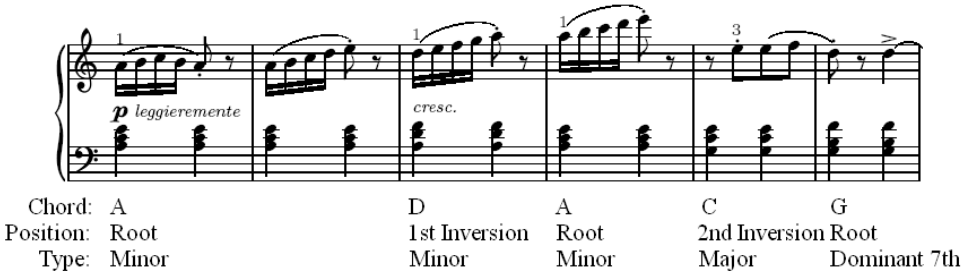


Figure 2.4: An excerpt from J. F. F. Burgmüller's l'Arabseque for piano, demonstrating chords and chord progressions.

Timbre is the musical feature that deals with the quality of a musical sound; for example, the timbre of a note played by a clarinet is different from the timbre of a note played by a trumpet. Also, the timbre of a note produced by plucking a violin string is different from a note produced by applying the bow to the violin. Therefore timbre is most often specified by the orchestration of a musical work which specifies the instruments involved, as well as by particular instructions on how to play notes on a certain instrument.

In general, instructions on how to perform the music can be classified as *editorial* features. Examples of such features include dynamic markings which indicate the volume at which the music should be played, fingerings to indicate which fingers should be involved in playing particular notes, and ornamentations. Music containing a vocal component often has lyrics, which are classified as *textual* features. Lastly, the *bibliographical* information concerning a musical work corresponds to musical metadata—this feature relates to information about a musical work, instead of its content.

The format of the musical data affects which musical features can be easily extracted from a musical work by the system. At one extreme, we can use a large amount of space to store complete representations of music, from which we can obtain information about all musical features. On the opposite end of the spectrum, an example of a sparse representation of music would be simply a series of pitches, without any temporal context. Here the tradeoff between robustness and scalability is apparent; more complete musical representations require much greater amounts of data storage than sparse musical representations. However, a robust music information retrieval system should be able to handle complex, flexible queries. Depending on the nature of the user queries, some degree of representational completeness is required.

2.5 Specification of the Music Pattern Matching Problem

The music pattern matching problem is to find the "best" match of a musical fragment, called a *pattern*, into a larger musical fragment called a *score*. The notion of what constitutes a "good" match varies with the approach, but generally some form of approximate matching is required to yield musically significant results. This is especially true when extending the problem to find the best k matches of the pattern into the score. Most previous approaches to the music pattern matching problem consider only the case where the pattern is monophonic; such monophonic patterns usually correspond to some part of a musical melody. The score generally corresponds to a complete musical work, which can be monophonic or polyphonic.

Our approach to this problem accommodates polyphonic patterns and scores. We also allow general weight models for approximate matching, which allows our algorithm to find various types of musically sensible matches. Our algorithm applies to symbolic representations of music, rather than audio representations of music. For more information about audio representations, see Foote [16]. While pattern matching using audio representations is possible, especially in the query by humming scenario (as in Shifrin et al. [47]), using a symbolic representation of musical notes as horizontal line segments provides a solid basis for several musically sensible ways of finding approximate matches of the pattern into the score. As we shall see in Chapter 5, there are also interesting relationships between the music pattern matching problem in this geometric format and certain problems in computational geometry.

Chapter 3

Previous Work

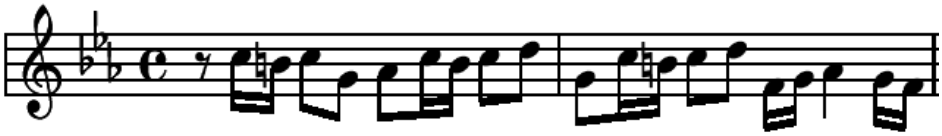
There have been many different approaches to music pattern matching based on symbolic representations of music. We first review methods that deal with monophonic music; these draw their inspiration from classical text information retrieval, and primarily use string representations of music. We then examine approaches to polyphonic music. When we have multiple notes overlapping in time, it seems more natural to represent music geometrically, rather than as a string.

3.1 Monophonic Symbolic Representations

The music pattern matching problem is much easier when restricted to a monophonic pattern and a monophonic score. Because monophonic music consists of a single voice, we can represent monophonic music as a single sequence of events through time. A simple representation could contain only the pitch information of each note—this can be seen in the first two string representations in Figure 3.1. Pitch contour and pitch interval representations track the pitch difference between each pair of consecutive notes. In order to accommodate information about note durations, we can construct a string based on an extended alphabet where each character is a (pitch, duration) pair. We can also have repeated occurrences of note pitches to denote a longer note, similar to a unary encoding of numbers.

3.1.1 String Matching and Edit Distance

Techniques for string matching are well established in the field of text information retrieval. Because monophonic music lends itself so well to a string representation, much work has been done to adapt string matching techniques for music information retrieval [29].



Note pitches: C B C G A♭ C B C D G C B C D F G A♭ G F

MIDI pitches: 72 71 72 67 68 72 71 72 74 67 72 71 72 74 65 67 68 67 65

Pitch contours: − + − + + − + + − + − + + − + + − −

Pitch intervals: −1 1 −5 1 4 −1 1 2 −7 5 −1 1 2 −9 2 1 −1 −2

(pitch, duration) pairs: (C, 1/16) (B, 1/16) (C, 1/8) (G, 1/8)
(A♭, 1/8) (C, 1/16) (B, 1/16) (C, 1/8) (D, 1/8)
(G, 1/8) (C, 1/16) (B, 1/16) (C, 1/8) (D, 1/8)
(F, 1/16) (G, 1/16) (A♭, 1/4) (G, 1/16) (F, 1/16)

Repeated unit time steps: C B C C G G A♭ A♭ C B C C D D
G G C B C C D D F G A♭ A♭ A♭ A♭ G F

Figure 3.1: A monophonic melody from J. S. Bach's Fugue in C minor, Well-Tempered Clavier Book I, followed by several possible string representations.

While exact string matching techniques such as the Boyer-Moore algorithm [4] are well-known, directly applying such techniques to the musical domain may miss interesting approximate matches due to musical variation. One such technique for approximate string matching involves the concept of edit distance.

In the textual setting for edit distance, we have two strings A and B over some alphabet, and wish to determine the edit distance between A and B, denoted as $ED(A, B)$. Typically there are three edit operations: insertion of a character, deletion of a character, and replacement of a character. In the most basic formulation, $ED(A, B)$ represents the minimum number of edit operations required to transform string A into string B. For example, $ED(kin, kiln) = 1$ by inserting the letter "l" into "kin", while $ED(dogs, cot) = 3$ by replacing the letters "d" and "g" with "c" and "t" respectively, and then deleting the letter "s". This basic form of edit distance is clearly a metric; that is, $ED(A, A) = 0$, $ED(A, B) > 0$ for any $A \neq B$, $ED(A, B) = ED(B, A)$, and $ED(A, C) \leq ED(A, B) + ED(B, C)$ for all strings A, B, and C.

Several extensions can be made to this basic edit distance model. The most common extension involves assigning positive costs c_{insert}, c_{delete}, and $c_{replace}$ to each of the three edit operations. While the basic edit distance model simply assigns costs of 1 to each edit operation, in some applications one type of edit operation may be more or less costly than another. Another option involves varying the cost of a particular edit operation based on the letters inserted, deleted, or replaced. This can make it "cheaper" to use certain letters over others.

The edit distance model allows for approximate string matching; if $ED(A, B) < ED(A, C)$, then according to that edit distance function we would classify string A as being more similar to string B than to string C. We can also examine how similar B is to A by looking at the value of $ED(A, B)$. We can calculate the edit distance by using dynamic programming techniques [52]. Variations on edit distance such as finding the longest common subsequence of two strings are heavily explored in other fields, such as bioinformatics [21]. In the context of bioinformatics, costs for insertions and deletions may depend in a more complex way on the length of the insertion or deletion.

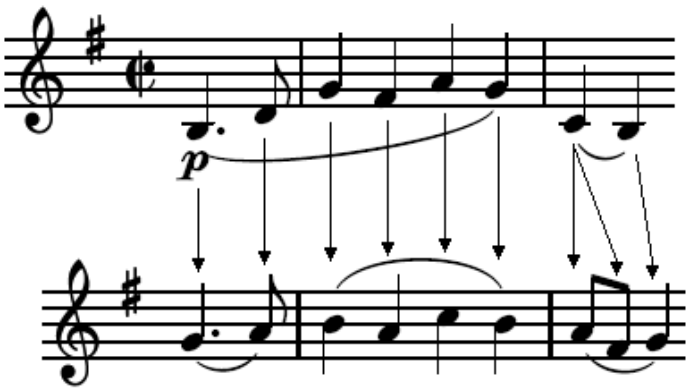
3.1.2 Musical Edit Distance

When we extend the edit distance model to music, we can have analogous edit operations such as inserting a note, deleting a note, or modifying the pitch of a note to transform one monophonic musical fragment into another. The measure of musical similarity is then defined by the minimum cost of the edit operations required to transform the pattern, with lower costs corresponding to greater similarity. Due to the temporal aspect of music, however, insertion and deletion of notes do not make musical sense. If we start with a monophonic musical pattern, and wish to match it into a larger musical score, we want to maintain the duration of the pattern, and inserting or deleting notes will change the duration of the pattern.

In 1990, Mongeau and Sankoff [37] presented a unique edit distance framework with two important modifications. First, they defined two additional edit operations that do not make sense for normal strings, but are useful when applied to a string representation of music. In addition to inserting, deleting, or modifying a note, they allowed the *consolidation* of multiple notes into one longer note, and the *fragmentation* of one note into multiple shorter notes. Secondly, this framework assigns different costs to the edit operation of modifying the pitch of a note depending on the relative pitches of the original note and the modified note. In Western music, some of these intervals between pairs of notes are more common than others, and Mongeau and Sankoff established a set of costs that reflected this. See Figure 3.2 for an example of how their technique can be used to compare two melodies.

In Figure 3.2, all notes in the first melody are linked to notes of similar duration in the second melody (with one fragmentation taking place for the second last note.) These replacement operations do not involve notes with different lengths, and therefore only the intervals between the notes determine how much each replaced note contributes to the overall edit distance. In the case of the fragmentation, the contributions of the intervals between the second last note and the fragmented notes are summed up. If any of these operations involved notes with different durations, non-zero w_{length} values would have contributed to the calculation of the edit distance.

This classic result is still significant due to the adaptation of the edit distance model to music, as well as the attempt to address the notion of similarity through interval-based



$w_{interval}$: 0.35 0.1 0.2 0.2 0.2 0.2 $(0.35 + 0.5)$ 0.35

$$ED(A, B) = \sum_{w_{interval}} = 2.45$$

Figure 3.2: Mongeau and Sankoff's musical edit distance between two excerpts of the second violin part to Haydn's "Emperor" String Quartet in C major, Op. 76 No. 3 2nd movement.

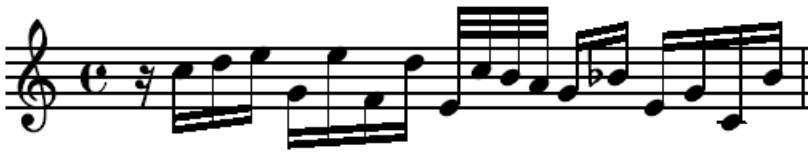
modification costs. We will present a similar method of handling approximate matching by using an interval-based weight model in our algorithm.

3.1.3 n-gram Techniques

Another approach based on string matching uses n-gram techniques [13]. Melodies are represented by minimal string representations denoting only the note pitches, without their durations. An interval-based string is obtained from the pitch-based string by assigning a letter to each interval between consecutive pitches, so that if the pitch-based string has length $k+1$, the interval-based string has length k. The letters used to represent particular intervals depend on a classification scheme.

Each classification scheme denotes an interval of 0 semitones with the letter "a". The least informative classification scheme, C3, simply denotes a negative interval of any magnitude with the letter "b", and a positive interval of any magnitude with the letter "B". Such a representation only gives information about whether we stay on the same note, move up to some other note, or move down to some other note. In contrast, a more informative classification scheme such as C15 denotes negative intervals of magnitude 1 to 6 with the letters "b" to "g", positive intervals of magnitude 1 to 6 with the letters "B" to "G", all negative intervals of magnitude 7 or greater with the letter "h", and all positive intervals of magnitude 7 or greater with the letter "H". While we can obtain

more information about the intervals from a C15 representation compared to a C3 representation, the additional letters used in the C15 representation require more storage space. See Figure 3.3.



MIDI pitches:	72	74	76	67	76	65	74	64	72	71	69	67	70	64	67	60	70
C3 representation:	B	B	b	B	b	B	b	B	b	b	b	B	b	B	b	B	
C15 representation:	C	C	h	H	h	H	h	H	b	c	c	D	g	D	h	H	

Figure 3.3: The first bar of the melody of J. S. Bach's Prelude in C major, Well-Tempered Clavier Book 2, followed by its MIDI pitch-based string representation, and C3 and C15 representations of the corresponding interval-based string. There are 17 notes in the fragment, and 16 corresponding pitch intervals.

Given the interval-based string of length k, we break the interval-based string into $n-k+1$ substrings called n-grams for $n < k$. See Figure 3.4. Downie and Nelson developed a music information retrieval system that stored the melodies of 9354 folksongs. By dividing the interval-based string associated with each melody into n-grams, the system can also divide queries into n-grams and find melodies that share common n-grams. In a sense, each n-gram corresponds to a musical "word" when we examine the analogous text approach. This approach is most useful for finding matches to a query from a large collection of melodies, because indexing techniques can be applied to find common n-grams quickly in that setting. In the music pattern matching setting where we have just one melody to match our query against, this method almost reduces to the brute force approach to string pattern matching, especially if there are few repeated n-grams in the melody.

C15 representation: C C h H h H h H b c

4-gram representation: [C C h H] [C h H h] [h H h H] [H h H h] [h H h H]
[H h H b] [h H b c]

5-gram representation: [C C h H h] [C h H h H] [h H h H h] [H h H h H]
[h H h H b] [H h H b c]

Figure 3.4: A C15 interval-based string, followed by its 4-gram and 5-gram representations.

To evaluate their system, Downie and Nelson examined five factors: the type of classification scheme used, the n-gram length, the user query length, whether the user query corresponded to the beginning of a melody or to the middle of a melody, and whether the user query had errors in it or not. An error-free user query would correspond to an exact match to some part of a melody, in this case. They concluded that leaving the intervals unclassified (with a distinct letter for each interval) was ideal. Also a longer n-gram length ($n = 6$) works better when there are no errors in the user queries, while a shorter n-gram length ($n = 4$) maximizes the fault tolerance of the system.

The length of the query is not a concern if the query is error-free; however, if errors occur in the user query then longer queries allow the system to perform better than for shorter queries. There was no difference in performance when the query corresponded to the beginning of a melody or not—this led to the conclusion that it is better to store entire melodies in musical databases, instead of only the beginnings of melodies. As noted previously, exact queries were much more easily matched on this system than queries containing one or more errors.

The n-gram approach is simple, and relates closely to analogous text information retrieval methods. For a large collection of monophonic music, a properly constructed n-gram based system can perform quite well, further highlighting the point that more complicated systems that deal with more complex representations should yield benefits beyond the capabilities of this simple system, to outweigh the costs associated with such complexity.

3.2 Polyphonic Symbolic Representations

When we remove the monophonic restriction on our original problem, we can immediately see that strings are not a natural choice to represent polyphonic music. A string is, by nature, one-dimensional—for monophonic representations, this dimension corresponded to time. We need another dimension when dealing with polyphony, to handle notes that overlap in time. Multi-track strings are examined below, followed by several geometric representations of music.

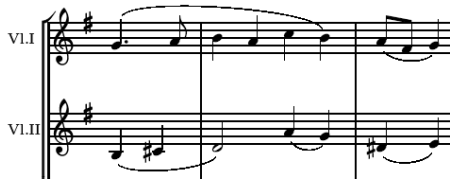
3.2.1 Multi-track Strings

Even if we consider the slightly more limiting problem which involves finding a monophonic pattern in a polyphonic score, it is difficult to directly apply string matching techniques. Despite this difficulty, there have been attempts to handle this problem using multi-track strings, for example by Lemström and Tarhio [30]. In this setting, polyphony is treated as the interaction of multiple monophonic voices, with a goal of finding matches of a monophonic pattern into this polyphonic score. Each monophonic voice in the score

is represented by a string, so the analogous problem in the text setting involves finding matches of a pattern string across parallel text strings.

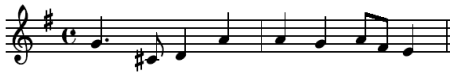
More precisely, the text T in the multi-track string setting consists of h parallel strings of length n, $s_i = s_{i,1}s_{i,2}\ldots\ s_{i,n}$ with $1 \leq i \leq h$, called *tracks*. A pattern string $p = p_1p_2\ldots\ p_m$ has an *occurrence across the tracks* h at j if $p_1 = s_{i_1,j}, p_2 = s_{i_2,j+1}, \ldots, p_m = s_{i_m,j+m-1}$ for some $i_1, i_2, \ldots, i_m \in \{1, 2, \ldots, h\}$. To formulate the equivalent problem in the music information retrieval domain, we must allow transpositions. Thus, given string representations of a monophonic musical pattern p and a score S consisting of h tracks $s_1, s_2, \ldots, s_h$ of length n, we wish to find all j's such that $p_1 = s_{i_1,j} + c, p_2 = s_{i_2,j+1} + c, \ldots, p_m = s_{i_m,j+m-1} + c$ for any constant c, and $i_1, i_2, \ldots, i_m \in \{1, 2, \ldots, h\}$.

The authors choose to only represent pitches in their string representations; an example based on MIDI pitches is shown in Figure 3.5 along with an example of matching a monophonic pattern into the polyphonic score. Because each voice in a polyphonic musical fragment cannot be guaranteed to have the same number of notes occurring at the same times, the λ symbol is used as a string character to indicate that no note is being played in a certain track at that moment in time. Therefore, the total number of characters in the score S, hn, is likely to be greater than the total number of notes in S; in the worst case the total number of notes in S can be $O(n)$, independent of h.



Track 1: **67 67** 69 71 **69** 72 71 **69 66** 67

Track 2: 59 61 **61 62** 62 **69 67** 63 63 **64**



Pattern: 67 67 61 62 69 69 67 69 66 64

Figure 3.5: An excerpt from the first and second violin parts to Haydn's "Emperor" String Quartet in C major, Op. 76 No. 3 2nd movement. The multi-track string representation is given, as well as a monophonic pattern that matches into the score (indicated by bold MIDI pitches).

Note that each character does not necessarily correspond to a single note, or to a uniform time step. Instead, each string has one character for every instant in time that a note changes in one of the tracks. This also means that the durations of the pattern notes may not exactly match the durations of the corresponding score notes.

The authors present a filtering algorithm with a running time exponential in the alphabet size to solve the multi-track string pattern matching problem.

Aside from the inherent limitation of searching for a monophonic pattern, this method has a few other drawbacks. Only pitch is explicitly represented in the strings; while the onset time of each note is implicitly represented via the position of a character in the string, as well as the use of the λ symbol, differences in note duration cannot be distinguished. The ability to track duration is essential to detect non-rudimentary musical matches. A way to address this issue would be to consider the smallest possible duration in the score as a unit of time, and have one character in each track for each unit of time as a unary encoding of duration. However, this would increase the total number of characters in the score significantly. Also, this framework is limited to exact pattern matching under transposition invariance. Modifications to the framework are needed to accommodate approximate pattern matching.

3.2.2 Transportation Distances

For polyphonic music, it is often more tractable to represent music in a geometric format, rather than as a string. One approach by Typke et al. uses weighted point sets in the plane, in combination with certain transportation distances [49]. This framework represents a musical note as a circle in the plane, with its horizontal coordinate representing the time at which the note begins, the vertical coordinate representing the pitch of the note, and the radius of the circle corresponding to the duration of the note. See Figure 3.6.

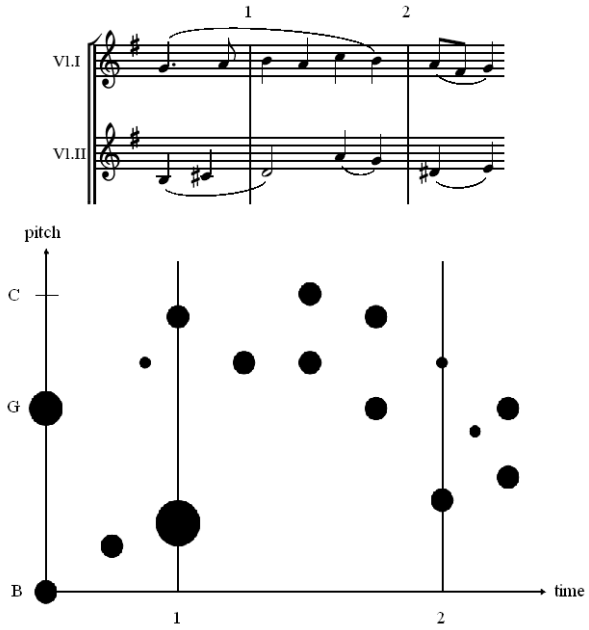


Figure 3.6: The weighted point set representation of the excerpt from Haydn's String Quartet.

The radius of a circle can naturally represent the duration of that note, which is similar to conventional Western musical notation where different ways of drawing a note correspond to different durations. This representation can be extended to have the radius represent other musical features. This model is also flexible enough to extend to additional weight components such as considering certain note stresses, which correspond to the location of a note in a bar. The transportation distances used to compute a match correspond to transferring the areas of the note circles in the pattern to corresponding note circles in the score. Imagine the pattern notes as piles of earth and the score notes as holes. We wish to transfer the earth into the holes with minimal effort; thus the main transportation distance featured in the paper is aptly called the Earth Mover's Distance. The Earth Mover's Distance between two weighted point sets can be computed by solving the corresponding linear programming problem.

The use of transportation distances to measure similarity was applied to a very large database of about half a million musical fragments. This generated improved results identifying musical fragments by previously anonymous composers, and grouping similar musical fragments together. Although the authors only experimented with monophonic music, the weighted point set model can be extended to polyphonic music since having multiple points with the same horizontal coordinate should not cause additional difficulty. A variant of the Earth Mover's Distance, called the Proportional Transportation Distance, satisfies the triangle inequality and therefore can be used to efficiently search large databases. While the Earth Mover's Distance allows for partial matching, the Proportional Transportation Distance does not.

Although transportation distances can yield meaningful partial matches by picking out melodic similarities obscured by additional notes or differing rhythm, false positives still occur. It is very likely that false positives will occur more frequently when comparing polyphonic music, as we shall see from the perspective of our algorithm. Further experiments using polyphonic music in this framework could potentially focus on aspects of transportation distances that can be modified to minimize certain types of false positives.

3.2.3 Multi-dimensional Point Sets

Another geometric approach represents polyphonic music as a multi-dimensional point set. By representing each note as a k-dimensional point, one can encode k quantifiable one-dimensional musical features by setting values for the appropriate point coordinates. The music pattern matching problem is then transformed into the geometric problem of finding a translation of a pattern represented as a k-dimensional point set of size m into a larger score consisting of n points in k dimensions.

Wiggins, Lemström, and Meredith present an algorithm to handle polyphonic music retrieval in this setting called SIA(M)ESE [53]. Their starting point is a pattern induction algorithm called SIA, which finds maximal repeated patterns in any k-dimensional set

of points. They present a music information retrieval task that focuses on finding the best match of a musical *template* (which can be a small pattern or a short tune) in a musical *dataset*, which is a musical database storing songs as k-dimensional point sets. This framework can be easily changed to address the music pattern matching problem.

This approach handles exact pattern matching, as well as some limited types of approximate matching. If no exact match of the pattern into the score exists, the SIA(M)ESE algorithm provides different translations for subsets of the pattern to exactly match into the score. One notion of what constitutes a good match uses the translation which exactly matches the longest time-contiguous subset of pattern notes into the score. Another type of approximate match can use the translation which exactly matches as much of the beginning and end of the pattern as possible into the score, under the rationale that the start and end of a musical phrase may be more memorable than the middle. See Figure 3.7.

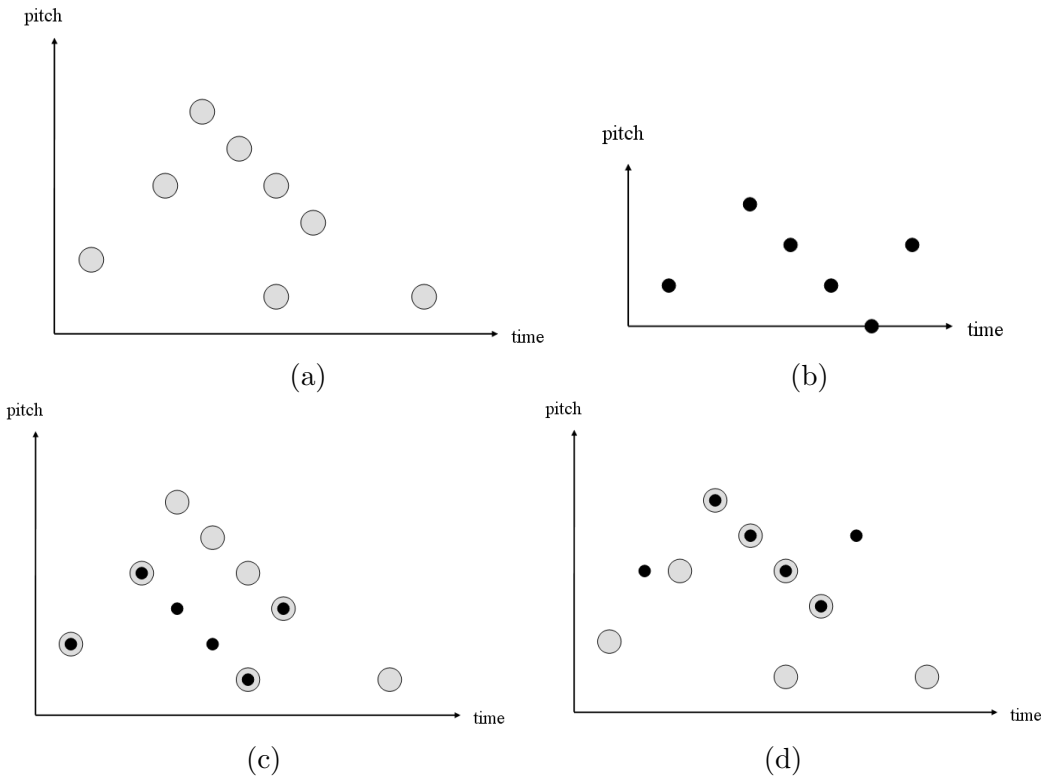


Figure 3.7: (a) Two-dimensional point set representation of a score; (b) two-dimensional point set representation of a pattern; (c) an approximate match of the pattern into the score that exactly matches the start and end of the pattern; (d) an approximate match of the pattern into the score that matches the longest time-contiguous subset of pattern notes into the score.

The SIA(M)ESE approach is good at finding matches of the pattern into the score where the number of pattern notes is less than the number of score notes in the part of the score that the pattern is translated to. The restriction of only considering approximate matches that exactly match one or more pattern notes into the score can ignore certain musically-sensible matches, especially if the dimensionality of the feature set under consideration is low. It is also difficult to deal with the problem of weighing certain musical features as either more important, or less important than others.

3.2.4 Line Segments

We use a geometric representation that has been used by many others, for example by Ukkonen et al. [51], in the Music Animation Machine [35], and by Brinkman and Mesiti [5]. Each note corresponds to a horizontal line segment in the plane, with the vertical axis measuring note pitches, and the horizontal axis measuring time. Therefore, the length of a horizontal line segment corresponds to the duration of the note, and the musical characteristics of pitch, time, and harmony are easily accessible using this format. See Figure 3.8.

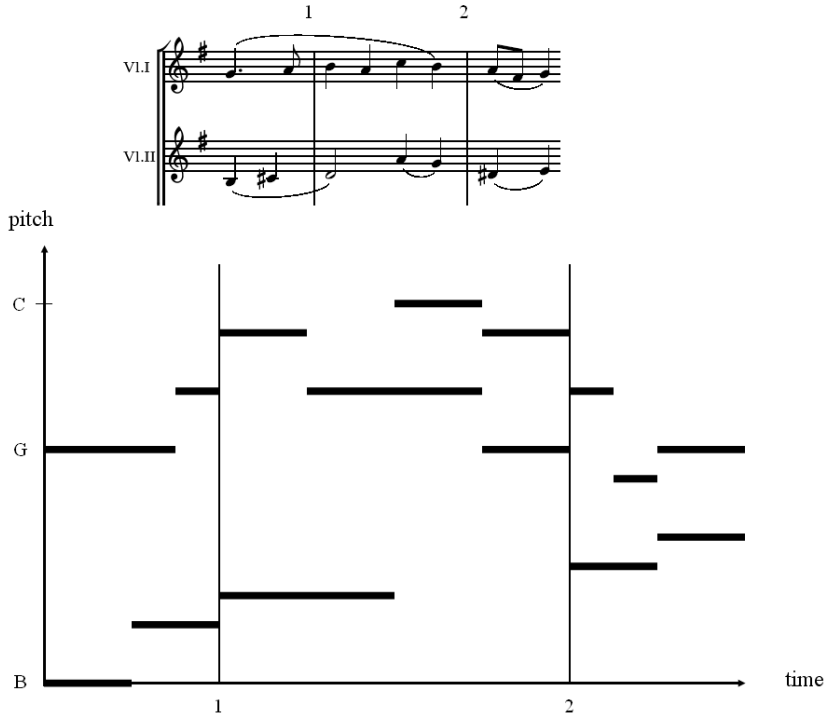
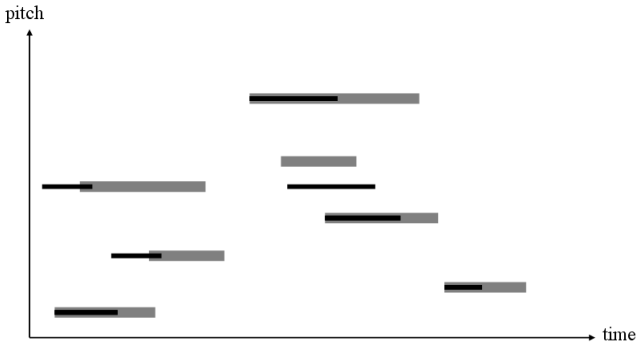


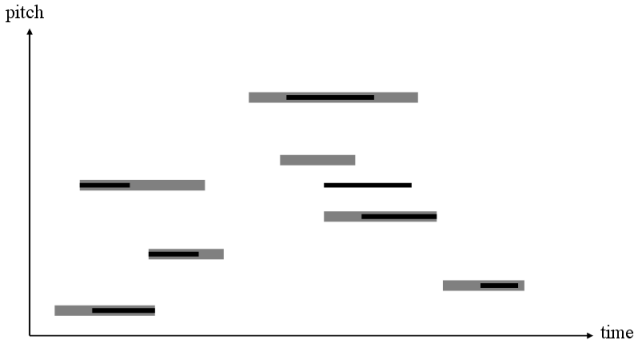
Figure 3.8: The line segment representation of the excerpt from Haydn's String Quartet.

Ukkonen et al. [51] provide algorithms to solve three types of music pattern matching problems. The first problem is to find translations of the pattern into the score where the start of each pattern note matches exactly with the start of a score note, in both time and pitch, where transpositions are allowed. A variant of this problem also requires note durations between the pattern and the score to be matched, which basically is a form of exact pattern matching. Because it is possible that no such translation exists for the first problem, their second problem is to find a translation of the pattern into the score that maximizes the number of note starts that match up between the pattern and the score.

The third problem is to find translations of the pattern into the score that result in the longest common shared time between pattern notes and score notes—we refer to this as the *longest common time* problem. Without the restriction of note starts having to match, this allows for a more interesting form of approximate pattern matching. See Figure 3.9. The problem of finding the longest common shared time also provided a great deal of motivation for the starting point of our algorithm.



(a)



(b)

Figure 3.9: (a) A match of a pattern (black line segments) into the score (grey line segments) that maximizes the number of note starts that match; (b) a match of a pattern into the score that maximizes the longest common time shared by pattern notes and score notes.

Another approach that has been applied to the line segment representation involves minimizing the area between two monophonic melodies. This concept was first explored by Ó Maidín [34], who represented monophonic music as a pitch-duration contour. See Figure 3.10. The notion of approximate matching explored here is more interesting than the longest common time problem in the previous algorithm.

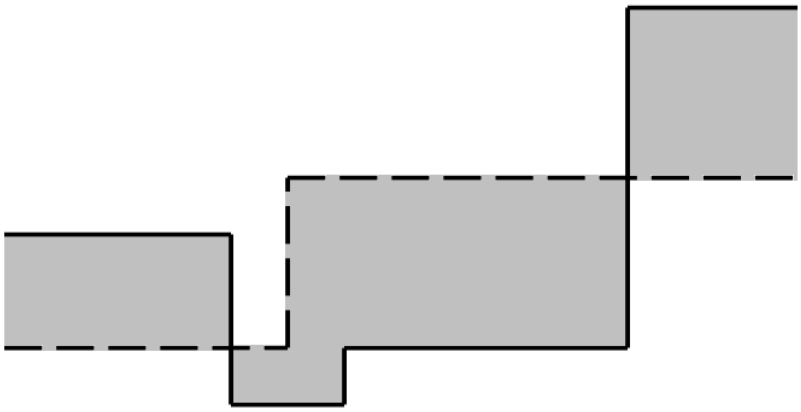


Figure 3.10: Two pitch-contour representations of monophonic music, and the area between them.

Francu and Nevill-Manning [17] expanded on this approach by using smaller units of time to quantize monophonic music with greater precision. This allows a set of horizontal alignments of the pattern with the score to be defined, and at each alignment, the best transposition that minimizes the area is stored. Eventually, the alignment and transposition combination that yields the minimum area is returned. Unfortunately, this algorithm has a very slow quadratic running time based on the temporal length of the pattern and the score, instead of the number of discrete notes in the pattern and the score.

Another approach deals with considering the pattern and the score to be cyclic monophonic melodies with period n, such as Indian ragas, as in Aloupis et al. [1]. The method used here can also be extended to the case where the monophonic pattern and score are non-cyclic. This contribution provides a much faster algorithm, running in $O(n^2 \log n)$ time in the cyclic melodies case, where n is the number of notes in each cyclic melody. Each melody is modeled as a series of horizontal line segments drawn around a cylinder, with vertical line segments connecting these as notes change. The objective is then to minimize the total area between the two melodies, to find the best match of one into the other. See Figure 3.11.

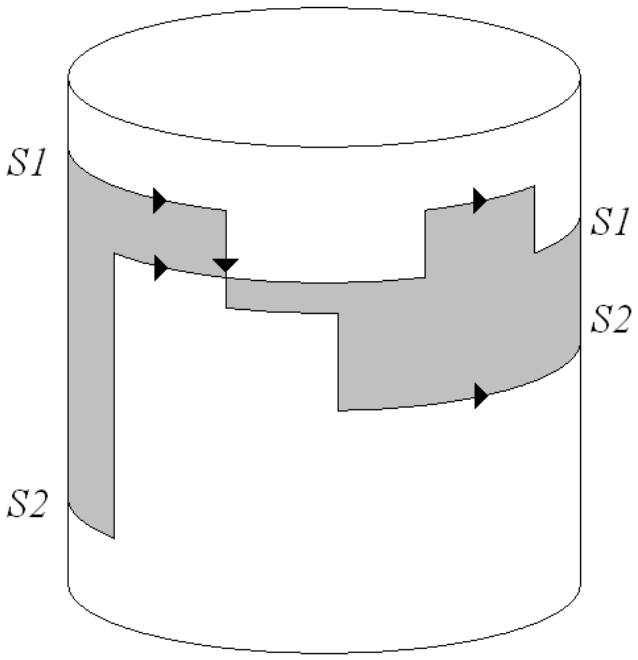


Figure 3.11: Matching cyclic melodies $S1$ and $S2$ according to the distance between note pitches. The "best" match involves shifting one of $S1$ or $S2$ so that the grey area is minimized.

This framework imposes a few limitations—unlike multi-dimensional point sets, a line segment representation cannot accommodate musical features other than pitch and duration. Also, multiple notes with the same pitch occurring at the same time are obscured in the visual representation, although a system can store separate notes with identical pitches, start times, and durations if needed. This also leads to another problem, where there is no difference between matching multiple pattern notes to a single score note at a particular time, and the case where there are two score notes with the same pitch, start time, and duration in place of the single score note.

Chapter 4

Algorithm

4.1 Overview

The music pattern matching problem that we are concerned with involves finding the "best match" of a pattern consisting of m notes into a score consisting of n notes. Both the pattern and the score may be polyphonic. We also have a *weight model* that determines what constitutes a good match. We now specify all details of the context in which we are addressing this problem.

By addressing the problem geometrically, we represent each note as a horizontal line segment in the plane. The horizontal axis represents time, while the vertical axis represents pitch. The horizontal coordinate of the left endpoint represents the time at which the note starts, while the horizontal coordinate of the right endpoint represents the time at which the note stops. Most schools of music utilize a discrete set of pitches, so we assume that line segments can only appear at d values on the vertical axis, where d is the size of the pitch set being used.

In our examples, we use the set of 128 MIDI pitches as our discrete pitch set. Our algorithm can be applied to other discrete pitch sets such as those based on scale degrees, or Hewlett's base-40 representation [22]. Larger pitch sets generally represent more information; a pitch set based on a base-12 representation cannot distinguish between the musical notes D♯ and E♭ (which have the same pitch, but are different in a musical sense), while Hewlett's base-40 representation can distinguish between these two notes. Therefore for a fixed pitch set we can consider d to be a constant factor; however, in the case of MIDI pitches it must be noted that 128 is a rather large constant. In the remainder of our analysis we continue to use d to allow the flexibility of different pitch sets, but note that in most cases it can be considered as a constant. Also, in many practical cases only a limited range of the 128 MIDI pitches is required.

Note that the d discrete pitch values need not be "evenly-spaced". Because we set

pitch-dependent weights according to the relationship between any two pitches, it does not matter if the d pitches occur in unit-length increments (as is the case for MIDI pitches, which correspond to musical semitones), or if the d pitches occur at uneven distances apart (as is the case for a discrete pitch set modeled on scale degrees, for example.)

We achieve a *match* of the pattern into the score by applying some translation $t = (x, y)$ to every note in the pattern. The horizontal component of t corresponds to starting the pattern at time x, while the vertical component of t corresponds to transposing the pattern by y pitches. A match of the pattern into the score must translate many pattern notes to overlap with score notes in time, in order to possibly be considered as a good match.

Our weight model determines what characteristics of the notes are compared in order to achieve a degree of similarity. A *pitch-based* weight model, for example, is defined by a weight function f which takes two note pitches as input and produces a number between 0 and 1 as output. This output is a measure of the similarity between the two input pitches, with a weight of 1 indicating the best possible match, and a weight of 0 indicating the worst possible match. Thus in a pitch-based weight model, the weight function is used to compare the pitches of a translated pattern note and a score note that overlap in time.

Each translated pattern note p overlaps with parts of zero or more score notes in time. The pitch-based weight contributed by p during a small enough time interval Δ can be expressed as $w_\Delta = \Delta \cdot max\{f(\pi(s), \pi(p))) : s$ is a score note overlapping with p for time interval $\Delta\}$, with $\pi(s)$ and $\pi(p)$ representing the pitches of notes s and p respectively. If zero notes overlap a pattern note in an interval, then that part of the pattern note contributes 0 to the total weight. We extend this to calculating how much weight each note p contributes by summing up the weights contributed by each part of p calculated as above, and then consider the total weight of the match to be the sum of contributed weight over all notes in the translated pattern. See Figure 4.1. More details about how the weight function works, and the properties of acceptable weight functions are given in Section 4.4.

To summarize, the "best match" of the pattern into the score is the translation that yields the best possible weight. We can address a different variant of the music pattern matching problem by retrieving the best k matches, in order to examine multiple occurrences of parts of the score that are similar to the pattern.

At its core, our algorithm takes the most basic approach to solving this problem: trying all possible translations of the pattern in relation to the score, and then returning the translation that yields the highest weight as the best match. The key realization is that we only have to check a discrete set of possible translations, of size $O(nmd)$; this will be proven and discussed in Section 4.5. We accomplish this in an efficient manner, with our algorithm running in $O(nm(d + \log m))$ time.

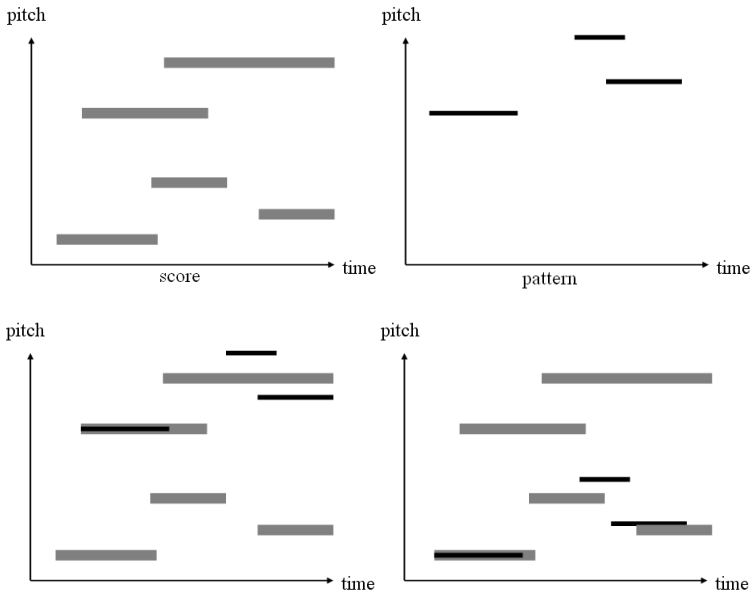
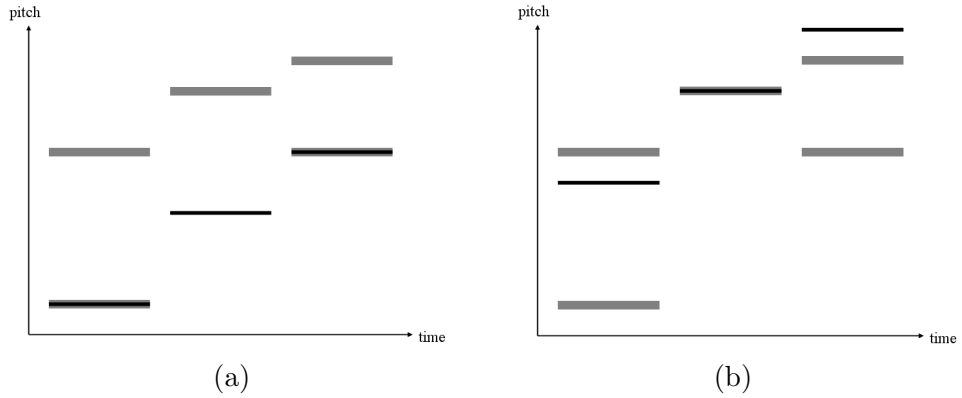


Figure 4.1: Two matches of a pattern (black line segments) into the score (grey line segments.) Each pattern note played during the horizontal interval $[t, t']$ is compared to score notes that occur during that same interval in order to calculate the weight contributed by each pattern note.

Previous algorithms that have comparable running times exist; for example, Ukkonen et al. [51] and Aloupis et al. [1] both achieve algorithms with running time $O(nm \log m)$. Although these running times do not depend on the size of the discrete pitch set as the running time of our algorithm does, these algorithms have restricted capabilities compared to our algorithm. In fact, as we will discuss in Section 4.4, we can perform the same type of pattern matching that these algorithms achieve by examining two special cases of our weight models.

As discussed in Subsection 3.2.4, the longest common time algorithm of Ukkonen et al. [51] defines the best match of the pattern into the score as the translation of the pattern that maximizes the total length of coincident pattern and score note fragments to address the third problem that they examine; see Figure 4.2(a). The algorithm of Aloupis et al. [1], on the other hand, defines the best match of the pattern into the score as the translation of the pattern that minimizes the total area between pattern and score notes that occur at the same time; see Figure 4.2(b). Their algorithm further requires that both the pattern and the score be monophonic, and that the score and pattern have the same duration.



(a) (b)

Figure 4.2: Two simple weight models yielding different best matches of the pattern into the score: (a) using the longest common time algorithm of Ukkonen et al., the best match is achieved by the total length of coincident pattern and score note fragments; (b) using the algorithm of Aloupis et al., the best match is achieved by minimizing the area between pattern and score notes.

Our algorithm handles polyphony in both the score and the pattern. While there have been other algorithms that have dealt with polyphony as seen in Section 3.2, the majority of previous music pattern matching algorithms focus purely on monophonic pattern matching, due to its close relationship to string pattern matching. Pattern matching in polyphonic music is important, because the majority of music is polyphonic, and often the interaction between multiple musical voices is what makes such music memorable. Thus finding musically sensible matches in polyphonic music can be very useful to a user.

More importantly, our algorithm allows a wide variety of general weight models (including two that accommodate the simple notions of a "best match" presented above). Previous algorithms tend to focus on one particular type of music pattern matching, as seen in Chapter 3. Our algorithm can handle many different ways of finding musically sensible matches by simply constructing an appropriate weight model.

We successfully apply our weight models to complex polyphonic music by using the discrete pitch set to define the weight of each possible pair of note pitches. Commonly used weight functions usually depend on the pitches of each of the two input notes to some extent. Although the additional time complexity that arises from using the discrete pitch set in this manner is undesirable, it is necessary to ensure that our algorithm is flexible enough to handle many different musically sensible weight models.

A more detailed account of how our algorithm works is given in Section 4.6; we give a brief overview here. We assume that the notes of the score and pattern are sorted by the time at which each note starts, in increasing order. If this is not the case then we sort both the score and the pattern at a cost of $O(n \log n + m \log m)$. Using the sorted score notes, we calculate up to $2n$ *time points*, which correspond to the horizontal positions

of the endpoints of each of the n score notes. We use these time points to determine the set of possible horizontal translations of the pattern into the score; each horizontal translation is achieved by aligning an endpoint of a pattern note with one of the time points, so there are up to $4nm$ horizontal translations. (We prove in Section 4.5 that only this set of horizontal translations need to be considered.)

We examine each of the horizontal translations in order. For each of the up to $4nm$ horizontal translations, we check each possible transposition of the pattern. Therefore at each step, we must update some information concerning the current translation, and then we must find the next translation. We move from one horizontal translation to the next by repeatedly extracting the minimum translation value from a heap of size $2m$, with corresponding pointers to each of the $2m$ pattern note endpoints. Finding the next translation therefore takes $\mathrm{O}(\log m)$ time.

We maintain information for each possible transposition of the pattern that can be updated efficiently at each horizontal translation. This allows us to easily calculate the weight for the current horizontal translation in constant time.

Because there are $\mathrm{O}(d)$ possibilities for the transposition of the pattern, and the information associated with each transposition is updated in constant time, we have $\mathrm{O}(d)$ work being done at each horizontal translation. Coupled with the $\mathrm{O}(\log m)$ time required to move from one of the $\mathrm{O}(nm)$ candidate translations to the next, we end up with an algorithm that runs in $\mathrm{O}(nm(d + \log m))$ time.

In terms of space, we store a constant amount of information for each of the d pitches, we also have the heap of $2m$ translation values, and we store a list of the $2n$ time points. Therefore we require at least $O(d+m+n)$ space. The maximum amount of space required is $O(dn)$, because we can easily run the algorithm using a $d \times n$ weight matrix to determine how much weight is contributed at different times in the score by pattern notes at certain pitches. We can actually calculate weights on the fly instead, which allows a reduction in the space complexity, but does introduce an additional factor of l to the running time, where l is the maximum polyphony of the score (that is, the largest number of score notes that are played simultaneously at any point in time.)

4.2 Comparison of Our Algorithm to Previous Work

We now provide a comparison of the capabilities and efficiency of our algorithm, compared to the previous results discussed in Chapter 3. We discuss the benefits and drawbacks of the techniques previously mentioned in Tables 4.1 and 4.2 below.

Approach	Efficiency	Type of Matching	Monophonic or Polyphonic
Basic string matching	$O(m+n)$	Exact matching	Monophonic
Edit distance	$O(nm)$	Limited approximate matching	Monophonic
Musical edit distance [37]	$O(nm)$	Approximate matching	Monophonic
n-grams [13]	$O(nm)$	Limited approximate matching	Monophonic
Multi-track strings [30]	Linear in n, exponential in alphabet size	Exact matching	Monophonic pattern, restricted polyphonic score
Transportation distances [49]	Polynomial in n and m	Limited approximate matching	Polyphonic
Multi-dimensional point sets [53]	$O(knm \log nm)$, k = dimensionality	Limited approximate matching	Polyphonic
Line segments [51, 17, 34, 1]	$O(nm \log m)$	Limited approximate matching	Monophonic (polyphonic score in some cases)
Our algorithm [33]	$O(nm(d+\log m))$	Approximate matching	Polyphonic

Table 4.1: Characteristics of previous work for music pattern matching.

The efficiency of our algorithm is $O(nm(d+\log m))$, where d is the size of the discrete pitch set that we are using. Algorithms used to solve the monophonic version of the music pattern matching problem tend to have running times that range between linear and quadratic, while previous algorithms for polyphonic music tend to be slower. Therefore the running time of our algorithm is comparable to the running times of previous algorithms for polyphonic music. As we will demonstrate, the manner in which our algorithm can find musically sensible approximate matches in polyphonic music is less limited than the methods of matching in these previous algorithms.

Approach	Benefits	Drawbacks
Basic string matching	Extremely fast	Only adequately deals with limited representations, no approximate matching, no polyphony
Edit distance	Well-established approach from previous string matching results, potential for more efficient heuristics used in other fields to be applied here	Only adequately deals with limited representations, very limited approximate matching, no polyphony
Musical edit distance [37]	Builds upon basic edit distance approach, uses edit operations that make more musical sense, uses interval-based weights that make more musical sense	No polyphony, edit operations still not completely ideal
n-grams [13]	Corresponds to matching words in text domain, simple method, works well for simple musical data such as folksongs, can be used to index large collections of music	No polyphony, only deals with pitch, not useful for complex music
Multi-track strings [30]	Uses established string matching techniques	Only deals with note pitch, unwieldy polyphonic representation
Transportation distances [49]	Intuitive geometric analogue to edit distance, natural representation of pitch and duration	Only deals with two musical features (usually pitch and duration), pattern must have same duration as the musical fragment that it is being compared to
Multi-dimensional point sets [53]	Flexible geometric representation, can take arbitrarily many musical features into account	Each approximate match must exactly match part of the pattern to the score, difficult to weigh importance of different musical features appropriately
Line segments [51, 17, 34, 1]	Natural representation of duration and pitch, can easily visualize if a match is "close" or not	Only deals with two musical features (usually pitch and duration)
Our algorithm [33]	General weight models for musically sensible pattern matching	Only deals with two musical features (usually pitch and duration)

Table 4.2: Benefits and drawbacks of previous work for music pattern matching.

Our algorithm combines the best elements of the musical edit distance and line segment approaches. We use the longest common time problem found in [51] as the basis for our simplest form of approximate pattern matching. Our algorithm also incorporates the distance-based weight found in [34, 17, 1] for a slightly more meaningful way of finding

approximate matches, adapted to the polyphonic setting. We then consider more general pitch-based weight models, taking inspiration from Mongeau and Sankoff's work on musical edit distance [37] to apply interval-based weights to the line segment representation of music.

Therefore our algorithm allows musically meaningful ways of pattern matching in polyphonic music, while still using a natural line segment representation. One noticeable drawback of our algorithm is that it only deals with the musical features of pitch and duration, but as seen in the literature, these two features are often more important than any other musical features that can be represented. Approaches using multi-dimensional point sets can take other useful musical features such as timbre into account.

4.3 Notation

In order to examine the details of our algorithm more precisely, we will introduce relevant notation and describe the uses of different variables. We also state the assumptions that we make about the input data, and the additional work needed if these assumptions are not met.

By representing a pattern or score note as a horizontal line segment, we identify three important quantities for a particular note s. We denote the *start time* of s as $\sigma(s)$; geometrically this corresponds to the horizontal coordinate of the left endpoint of the line segment. Similarly, the *stop time* of s is denoted by $\tau(s)$, corresponding to the horizontal coordinate of the right endpoint of the line segment. Finally, we denote the pitch of the note as $\pi(s)$, which matches the vertical coordinate of the line segment. See Figure 4.3.

We require that both the set of score notes and the set of pattern notes are sorted in increasing order of start time. This is generally true for input data obtained from a MIDI file, or some other organized music file format. If the data is unsorted, then we sort both the score and the pattern in $O(n \log n + m \log m)$ time before we proceed with our algorithm.

The set of time points is defined by all distinct $\sigma(s)$ and $\tau(s)$ values, over all score notes s. Since we have n score notes, there are at most $2n$ time points; note that if there are exactly $2n$ time points, then the score must be monophonic with rests between each pair of successive notes, so in most cases the number of time points is much fewer than $2n$. For the purposes of our algorithm, we require a sorted list of time points. Because the score notes are sorted by increasing $\sigma(s)$, we can obtain a sorted list of time points by scanning through the list of score notes once, keeping a heap of $\tau(s)$ values for the notes that are currently being played. This takes time $O(n \log l)$ where l represents the maximum polyphony of the score. l is assumed to be bounded by a constant, as in almost all cases l is significantly smaller than d.

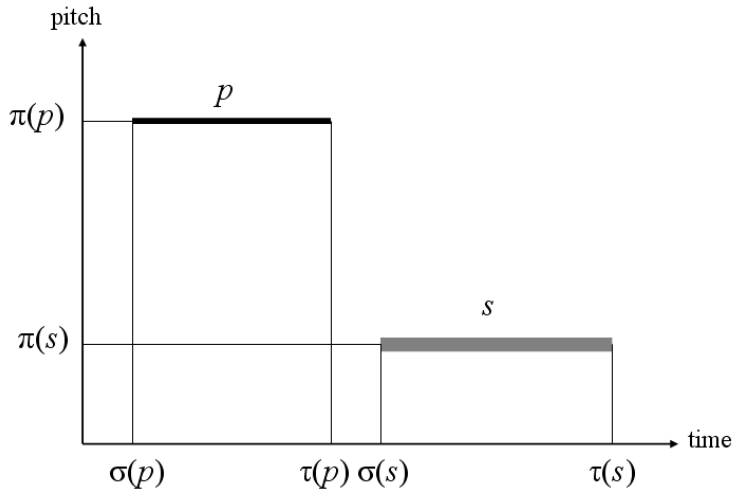


Figure 4.3: Pictorial representation of notation described in this chapter.

4.4 Weight Models

We use our weight model to determine what constitutes a good match of the pattern into the score. Our weight model is defined by a weight function f that assigns numerical weights based on differences between translated pattern notes and the score. The weight function is additive, in that the weight of a particular match is simply the sum of the weights contributed by each translated pattern note. Note that there is no additional cost required to translate the pattern; only the differences between the current translation of the pattern and the score contribute to the weight of a match.

Given our geometric representation of notes as line segments, we note that our weight model must compute weights based on note pitches, note durations, or a combination of both. In this section we focus on pitch-based weight models. Our algorithm is designed to handle pitch-based weight models efficiently, as comparing note pitches forms the basis of most practical approaches to music pattern matching.

We consider pattern notes of longer duration to be more important than pattern notes of shorter duration; this approach has been used in the past, as in the longest common time problem of Ukkonen et al. [51]. Therefore our weight model has longer pattern notes contribute more weight than shorter pattern notes. To handle this, we set the weight of a translated pattern note p to be proportional to its duration, $\tau(p) - \sigma(p)$. Thus if a single score note s is matched against p for the entire duration of p, the weight contributed by p is equal to $(\tau(p) - \sigma(p))f(\pi(s), \pi(p))$, where f is a pitch-based weight function. More generally, if s overlaps with p for a duration δ that is less than $\tau(p) - \sigma(p)$, then the weight contributed by that portion of p of duration δ is equal to $\delta f(\pi(s), \pi(p))$. If a

portion of p overlaps with no score notes (that is, a rest in the score), then that portion of p contributes zero weight by default.

It is possible for portions of a single translated pattern note to match to different score notes in a monophonic score. In this case, the total weight contributed by the pattern note is the proportionally-weighted sum of the weight contributed by each portion of the pattern note and the corresponding score note. This framework can handle Mongeau and Sankoff's [37] musical edit operation of "fragmentation", discussed in Subsection 3.1.2. The converse "consolidation" operation is dealt with when a sequence of multiple pattern notes match to the same score note. Visual examples of determining how portions of pattern notes can be matched into the score are presented in Figure 4.4.

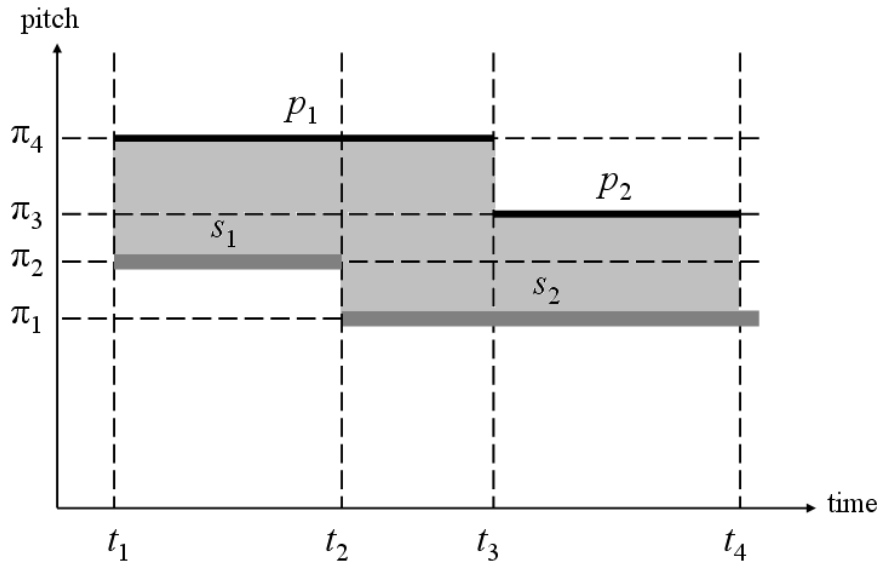


Figure 4.4: Computing the weight when both the score and pattern are monophonic. The weight in this example is $(t_2 - t_1)f(\pi_2, \pi_4) + (t_3 - t_2)f(\pi_1, \pi_4) + (t_4 - t_3)f(\pi_1, \pi_3)$.

More generally, if we have a polyphonic score, we can match each portion of a pattern note against the portion of a score note (occurring at the same time) that yields the best weight. An example of calculating the best weight of a match in the polyphonic setting is presented in Figure 4.5.

Our simplest weight model is based on the longest common time approach of Ukkonen et al. [51]. We call this the {0, 1}-weight model, with the corresponding weight function $f(\pi(s), \pi(p))$ taking the value 1 if $\pi(s) = \pi(p)$, and 0 otherwise. Thus the longest common time match of the pattern into the score is the translation that yields the maximum weight. While this is a form of approximate pattern matching that can deal well with inexact note starts, it does not capture any approximate similarity in terms of note pitches, because pattern and score note pitches must match exactly in order to yield a non-zero weight.

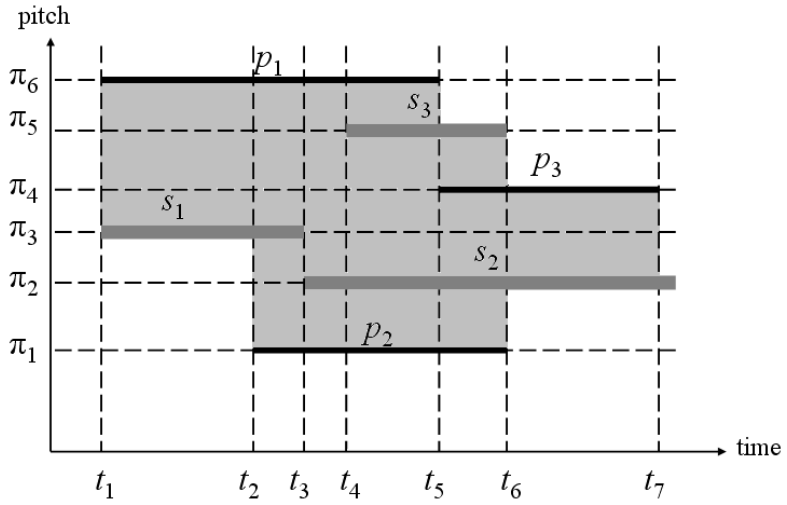


Figure 4.5: Computing the weight when both the score and pattern are polyphonic. The weight contributed by p_1 is $(t_3 - t_1)f(\pi_3, \pi_6) + (t_4 - t_3)f(\pi_2, \pi_6) + (t_5 - t_4)\max(f(\pi_2, \pi_6), f(\pi_5, \pi_6))$; the weight contributed by p_2 is $(t_3 - t_2)f(\pi_1, \pi_3) + (t_4 - t_3)f(\pi_1, \pi_2) + (t_6 - t_4)\max(f(\pi_1, \pi_2), f(\pi_1, \pi_5))$; the weight contributed by p_3 is $(t_6 - t_5)\max(f(\pi_2, \pi_4), f(\pi_4, \pi_5)) + (t_7 - t_6)f(\pi_2, \pi_4)$. The total weight in this example is the sum of the above three values.

We would like to use a more sophisticated weight model, where pattern notes that have different pitches than score notes occurring at the same time can affect the quality of a match.

The distance-based weight model is adapted from Aloupis et al. [1], with the most similar match of the pattern into the score corresponding to the translation of the pattern that yields the minimum area between pattern notes and score notes. In this case, if the weight of the match corresponds to the area between the pattern and the score, then lower weights correspond to more similar matches, with a weight of 0 corresponding to an exact pitch-based match of the pattern into the score. Thus the best match of the pattern into the score is the translation that yields the *minimum* weight. This weight model makes a lot of sense from a geometric standpoint, as it seems logical to classify notes that are closer together (in terms of pitch) as more similar than notes that are farther apart. However, this is not always true in the musical context.

Therefore we have developed an interval-based weight model, similar to that used in Mongeau and Sankoff's [37] edit-distance framework. Recall from Section 2.4 that an interval is the difference, in pitches, between two notes. The reason why the distance-based weight model does not always make sense is because certain intervals are more or less common than others, according to the tenets of music theory. This leads us to define $f(\pi(s), \pi(p))$ to depend on the interval between $\pi(s)$ and $\pi(p)$ according to common

characteristics of music theory. For example, the common musical interval of a perfect 5^{th} (7 semitones) should receive a better weight than the dissonant interval of an augmented 4^{th} (6 semitones), even though the latter interval would receive a better weight in the distance-based weight model. As in the {0, 1}-weight model, we let $f(\pi(s), \pi(p)) = 1$ if $\pi(s) = \pi(p)$. All other intervals receive a non-negative weight less than 1, and the best match of the pattern into the score is the translation that yields the *maximum* weight. A simple example using this weight model is presented in Figure 4.6.

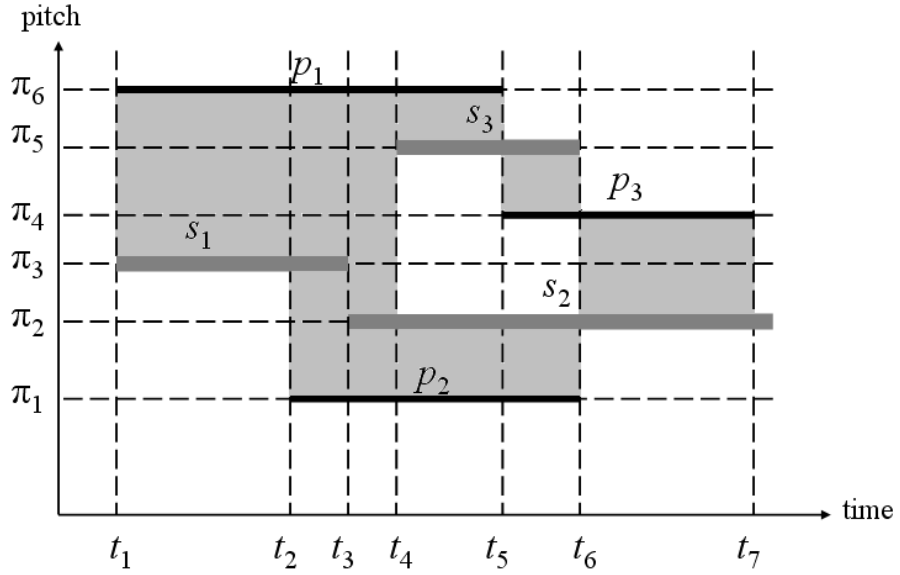


Figure 4.6: Calculating the weight of a match under an interval-based weight model. In this example we have $f(\pi_3, \pi_6) = 0.4$, $f(\pi_2, \pi_6) = 0.6$, $f(\pi_5, \pi_6) = 0.9$, $f(\pi_1, \pi_3) = 0.7$, $f(\pi_1, \pi_2) = 0.8$, $f(\pi_1, \pi_5) = 0.2$, $f(\pi_4, \pi_5) = 0.8$, and $f(\pi_2, \pi_4) = 0.7$. Thus the weight contributed by p_1 is $(t_3 - t_1)0.4 + (t_4 - t_3)0.6 + (t_5 - t_4)0.9$, the weight contributed by p_2 is $(t_3 - t_2)0.7 + (t_4 - t_3)0.8 + (t_6 - t_4)0.8$, the weight contributed by p_3 is $(t_6 - t_5)0.8 + (t_7 - t_6)0.7$, and the total interval-based weight of this match is the sum of these three values.

It is possible to consider other note characteristics in our weight models if we modify our representation of the notes. For example, our weight model can compare relative volumes of two notes (and hence handle musical dynamics) if we track the volume of each note, as well as the pitch, start time, and stop time. If we have information on the durations and positions of musical bars in the score, then we can assign weights based on how the starts of notes line up with particular beats in the bar, as certain beats are stressed more or less than others. In cases where we need to track multiple note characteristics simultaneously (for example, note pitch and note volume) or when we track a note characteristic other than pitch, modifications will have to be made to our algorithm that will increase the running time; further details are given in Section 4.7.

4.5 Correctness

The key towards making our algorithm efficient is finding a bounded set of candidate translations of the pattern in the score. With an appropriate weight model as described in the previous section, our algorithm can update the information required to calculate the current weight in constant time when moving from one candidate translation to the next. In fact, the weight models presented in the previous section depend on a bounded set of candidate translations of size $O(nmd)$, and we proceed to prove why this occurs.

Recall from Section 4.1 that we have up to $2n$ *time points* that correspond to the horizontal positions of the endpoints of each of the n score notes. We will prove that there must be an optimum match of the pattern into the score such that at least one pattern note endpoint coincides with a time point. In particular, consider the grid formed by the d pitches along the vertical axis, and the time points along the horizontal axis. See Figure 4.7.

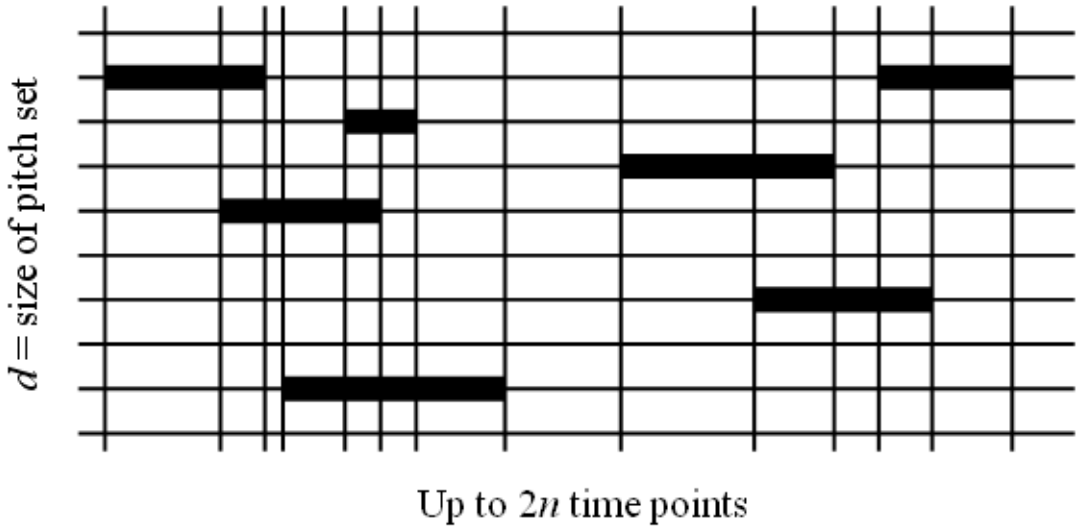


Figure 4.7: The grid defined by a score and the discrete pitch set.

Since there are at most $2n$ time points, there are at most $2dn$ points on the grid. Since there are $2m$ pattern end points, the size of the bounded set of candidate translations that we must check is at most $4dnm$.

Lemma 1 *For a pitch-based weight model as described in Section 4.4, there will be an optimum match of the pattern into the score such that at least one translated pattern note starts or stops at one of the grid points.*

Proof: Consider a translation of the pattern in the score that produces an optimum match. This translation must match note pitches to the grid, but suppose that no translated pattern note endpoint lies on one of the grid points.

Let ϵ denote the minimum horizontal translation we can apply to the translated pattern, such that shifting the translated pattern left or right by ϵ matches the start or end of a pattern note to a time point. If we examine a single translated pattern note p, we see that shifting p by ϵ', where $-\epsilon \leq \epsilon' \leq \epsilon$ does not affect the score notes that p matches to. Only the length of the portions of p that match to different score notes changes by ϵ'; see Figure 4.8.

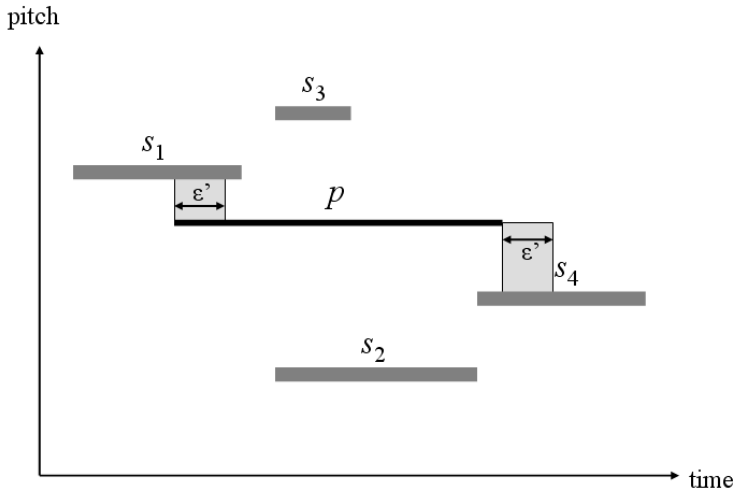


Figure 4.8: Shifting right by ϵ' changes the weight contributed by pattern note p by $\epsilon'(f(\pi(p), \pi(s_4)) - f(\pi(p), \pi(s_1)))$. Shifting left by ϵ' changes the weight contributed by pattern note p by $\epsilon'(f(\pi(p), \pi(s_1)) - f(\pi(p), \pi(s_4)))$.

Denote the original translation of the pattern into the score as $t = (x, y)$, and for ϵ' as defined above let $t_{left} = (x - \epsilon', y)$ and $t_{right} = (x + \epsilon', y)$. Let the weights associated with the translations t, t_{left}, and t_{right} be M, M_{left}, and M_{right} respectively. Because moving from translation t_{left} to t to t_{right} does not affect the score notes that p matches to, it must be the case that $M - M_{left} = M_{right} - M$. Therefore, without loss of generality, if $M_{left} < M$, we must also have that $M < M_{right}$. But t is a translation that yields an optimum match, which means that $M \geq M_{left}$ and $M \geq M_{right}$. So for $M - M_{left} = M_{right} - M$ to be satisfied, we must have $M = M_{left} = M_{right}$ with t_{left} and t_{right} being translations that also result in optimum matches. See Figure 4.9.

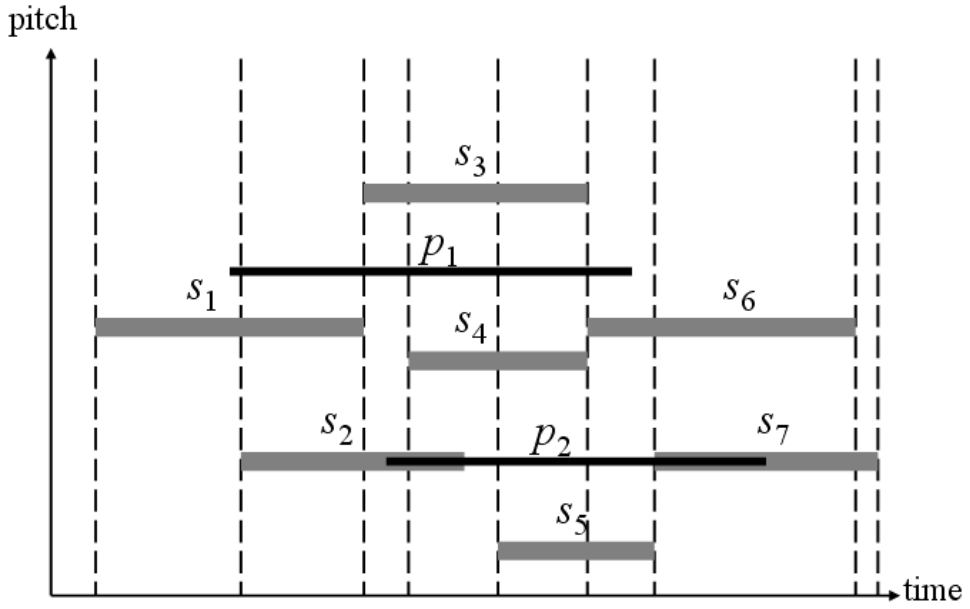


Figure 4.9: Assuming that $f(\pi(p_1), \pi(s_5)) < f(\pi(p_1), \pi(s_6))$, t_{left} and t_{right} are also optimum matches, since $\pi(s_1) = \pi(s_6)$ and $\pi(s_2) = \pi(s_7)$ in this example.

More precisely, for a pattern note p as above there must exist a value $\delta(p)$ such that $\epsilon'\delta(p) = M - M_{left} = M_{right} - M$. If the start of p matches to score note s_σ, the end of p matches to score note s_τ, and p has been transposed to pitch π, then $\delta(p) = f(\pi(s_\tau), \pi) - f(\pi(s_\sigma), \pi)$. The change in weight over all pattern notes is $\epsilon' \sum_p \delta(p)$. Note that $\sum_p \delta(p)$ cannot be positive or negative, since then a positive or negative ϵ' would increase the weight, and the original translation t would not yield the optimum weight.

Therefore $\sum_p \delta(p) = 0$, and we can shift the pattern from t a small distance to the left or right without having the total weight change. If we shift horizontally by ϵ and stop when a pattern note starts or stops at a grid point, the weight of the resulting translation will still be optimum.

Thus we have an optimum match on the grid. □

In some special cases, we can achieve a tighter upper bound on the size of our set of candidate translations of $4nm$, which essentially removes the dependence on the size of the pitch-set. If we find ourselves in a situation where any optimum translation must have at least one pattern note starting or stopping at the same time that a score note of the same pitch starts or stops, then our upper bound on the number of candidate translations is reduced to $4nm$. This bound holds for weight models that measure the exact overlap of pattern and score notes, such as our $\{0, 1\}$-weight model. This also holds

for the distance-based weight model with cyclic monophonic melodies as described in Aloupis et al. [1]. Note, however, that we cannot achieve this tighter upper bound when dealing with non-cyclic music as is generally the case when we use our distance-based weight model.

4.6 Algorithm and Analysis

Our algorithm can be divided into three distinct stages. We will first present some additional notation, and then give a detailed description of the entire algorithm. We will then more closely analyze the non-trivial aspects of our algorithm, and provide corresponding blocks of pseudocode where warranted.

Recall that the input to our algorithm consists of a score S and a pattern P, where S consists of a list of n triples of the form $(\sigma(s), \tau(s), \pi(s))$ ordered by increasing $\sigma(s)$, and P consists of a list of m triples of the form $(\sigma(p), \tau(p), \pi(p))$ ordered by increasing $\sigma(p)$. We are also given a weight function f which outputs a weight value for each pair of pitches (π_1, π_2). We denote a translation of the pattern into the score as (t, π) where t is the horizontal translation of the pattern in time, while π is the vertical translation of the pattern in pitch.

Given S, P, and f, our algorithm determines the translation (t_{opt}, π_{opt}) such that when P is translated by (t_{opt}, π_{opt}) in S, the translated pattern yields the optimum weight using the weight function f. In order to find the optimum translation, we try each of the $O(nmd)$ possible candidate translations defined in the previous section.

4.6.1 Detailed Description

Before we enter the main part of our algorithm, we must obtain an ordered list U of time points. Recall that each distinct $\sigma(s)$ and $\tau(s)$ value corresponds to one time point, and therefore we have up to $2n$ time points. Recall our assumption that notes of S are ordered by increasing $\sigma(s)$ values. We therefore have an ordered list of starting time points. We obtain an ordered list of stopping time points as follows: we examine each note $s \in S$ in order, adding $\tau(s)$ to a heap. Before we add $\tau(s)$ to the heap, we extract all heap values $\tau(s')$ such that $\tau(s') \leq \sigma(s)$. This reduces the maximum size of the heap from n to l, the maximum polyphony of the score. The heap values, in order of extraction, form the ordered list of stopping time points. At this point, we merge the two ordered lists of starting time points and stopping time points to obtain U, the ordered list of all time points.

The first stage of our algorithm precomputes the weight values to be applied to translated pattern notes. Our simplest approach stores weight values in a weight matrix W with one row for each of the pitches in the discrete pitch set, and one column for each of

the time points except for the last one. Therefore W has size at most $d \times (2n-1)$. We define $W(\pi, u)$ as the weight to be applied to the portion of any pattern note between time point u and the next time point u', assuming it is transposed to pitch π. Therefore for such a pattern note, the portion of it lying between u and u' contributes $d \times W(\pi, u)$ to the total weight of a match, where d is the duration of the portion of the translated pattern note lying between u and u'. More formally, $W(\pi, u) = \max\{f(\pi(s), \pi) : s$ is a score note being played during time interval $(u, u')\}$.

We compute all values of W by examining each score note s, and then updating $W(\pi, u)$ as π ranges through the d pitch values, and u ranges through the time points from $\sigma(s)$ up to, but not including, $\tau(s)$.

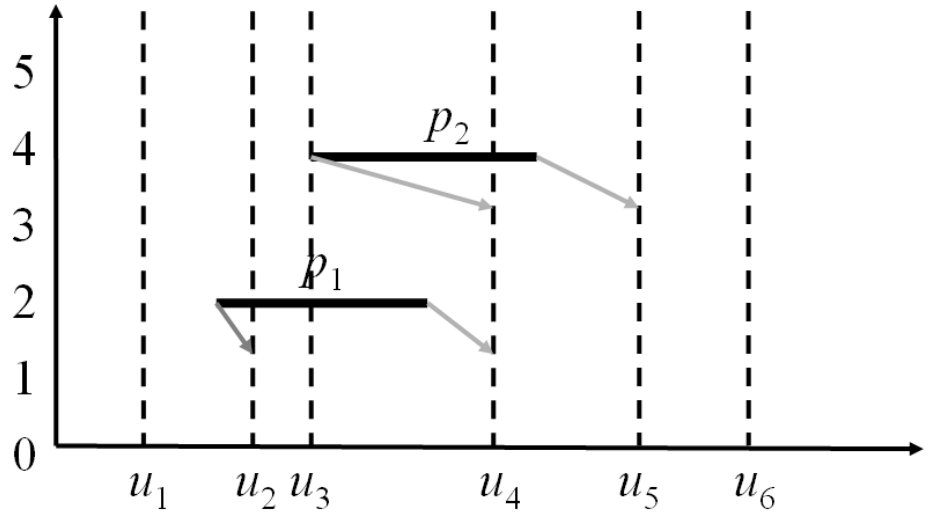
The second stage of our algorithm initializes our system for moving efficiently from one horizontal translation to the next, in order. A horizontal translation that results in the start or the end of a pattern note lining up with a time point is called an *event*. In particular, for time point u and pattern note p, a translation of $t = u - \sigma(p)$ corresponds to an event that matches the start of p to u. Similarly, a translation of $t = u - \tau(p)$ corresponds to an event as it matches the end of pattern note p to time point u. We examine each event in the order of their horizontal translation values. It is often the case that multiple time events have the same horizontal translation value, but we handle such events one at a time for book-keeping purposes.

For each pattern note p, we maintain two pointers p_σ and p_τ into the list of time points U. Pointer p_σ points to the smallest time point that the start of note p has not yet visited, while p_τ points to the smallest time point that the end of note p has not yet visited. Therefore when the start of p reaches the time point u corresponding to p_σ, that horizontal translation $t = u - \sigma(p)$ is an event. So before the start of p reaches time point u, the next event associated with pointer p_σ is the horizontal translation t. We define the next event for each of the other $2m-1$ pointers similarly.

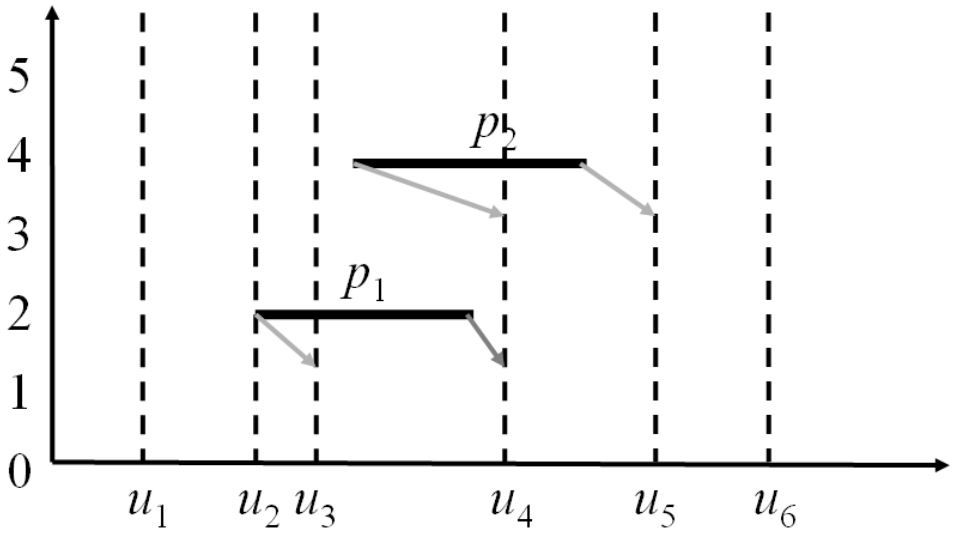
We store the $2m$ horizontal translation values that correspond to these next $2m$ events in heap *NextEvent*. This allows us to find our next event by extracting the minimum value from the heap, setting the associated pointer to point to the next time point u' in U, and inserting the new horizontal translation associated with that pointer back into our heap. Since each of the $2m$ pointers makes one pass through U, there are at most $4nm$ events. See Figure 4.10.

The third and final stage of our algorithm uses heap *NextEvent* to move from one horizontal translation to the next, and will efficiently update information needed to compute the weight of a match for each possible transposition π of the pattern. There are actually $2d-1$ possible transpositions of the pattern to explore assuming the set of discrete pitches ranges from 0 to $d-1$. By using π to run through $2d-1$ possible transpositions of the pattern from $-(d-1)$ to $d-1$, we handle extreme cases that could match a pattern note at pitch $d-1$ to a score note at pitch 0, or vice versa.

Let the horizontal translation associated with the current time event be t. For each



(a)



(b)

Figure 4.10: Moving from one event to the next: score notes removed for clarity. (a) At event t_0, the start of p_2 has reached time point u_3, so $t_0 = u_3 - \sigma(p_2)$. Therefore pointer $(p_2)_\sigma$ is updated to point to time point u_4. Now $(p_1)_\sigma$ points to time point u_2. The next event is $t_1 = u_2 - \sigma(p_1)$, corresponding to pointer $(p_1)_\sigma$. (b) After reaching t_1, $(p_1)_\sigma$ is updated to point to time point u_3. Note that $(p_1)_\tau$ points to time point u_4. The next event is $t_2 = u_4 - \tau(p_1)$, corresponding to pointer $(p_1)_\tau$.

transposition π, we maintain the weight of the translation (t, π), as well as other book-keeping information that allows us to update the weight in constant time. In particular, we maintain a value $\delta_\pi(p)$ for each value of π, and each pattern note p. Value $\delta_\pi(p)$ has the property that shifting the pattern horizontally by some small value ϵ (resulting in horizontal translation $t + \epsilon$) changes p's contribution to the weight of the match at transposition value π by $\epsilon\delta_\pi(p)$. We use ν to denote the pitch of the translated pattern note p, thus $\nu = \pi(p) + \pi$. Thus for a particular pattern note p, we have that $\delta_\pi(p) = W(\nu, prev(p_\tau)) - W(\nu, prev(p_\sigma))$ since the start of note p has passed the time point corresponding to $prev(p_\sigma)$ while the end of note p has passed the time point corresponding to $prev(p_\tau)$.

Two other book-keeping values are maintained for each transposition π; w_π stores the weight of the pattern translated by (t, π), while Δ_π is defined to equal $\sum_{p \in P} \delta_\pi(p)$. Note that if we shift the pattern horizontally by ϵ, then w_π changes by $\epsilon\Delta_\pi$. Therefore we update w_π, $\delta_\pi(p)$, and Δ_π, where p is the pattern note associated with the current event.

Suppose that the horizontal translation associated with the previous event is x, where it is possible that $t = x$. Then at the current event we first update w_π, by setting $w_\pi \leftarrow w_\pi + (t - x)\Delta_\pi$. We update w_π before Δ_π because Δ_π relates to the change in weight upon a small horizontal translation beyond t, that is $t \leftarrow t + \epsilon$.

In order to update $\delta_\pi(p)$ and Δ_π, we define a value δ_π as follows. If the current event corresponds to the start of p, then we define *nextWeight* to be equal to $W(\nu, p_\sigma)$ if p_σ is not the last time point in U, and 0 otherwise (since $W(\nu, p_\sigma)$ is undefined if p_σ is the last time point in U.) We also define *prevWeight* to be equal to $W(\nu, prev(p_\sigma))$ if p_σ is not pointing to the first time point in U, and 0 otherwise. We define *nextWeight* and *prevWeight* in the same way when the current event corresponds to the end of p, replacing $p\sigma$ with p_τ.

If the current event corresponds to the start of p, then $\delta_\pi = prevWeight - nextWeight$. If the current event corresponds to the end of p instead, then $\delta_\pi = nextWeight - prevWeight$. We can then update $\delta_\pi(p)$ and Δ_π as follows: $\delta_\pi(p) \leftarrow \delta_\pi(p) + \delta_\pi$, and $\Delta_\pi \leftarrow \Delta_\pi + \delta_\pi$.

We perform these updates at each event, and also maintain the translation that results in the best match seen so far. After checking all events, our algorithm returns the best weight value and the translation that achieves it.

4.6.2 Pseudocode and Running Time

As detailed in Section 4.3, assuming that we start off with sorted lists of score and pattern notes, it takes $O(n \log l)$ time to construct the ordered list U of distinct time points, where l is the maximum polyphony of the score. Therefore the preliminary preprocessing for our algorithm takes $O(n \log l)$ time.

Pseudocode for the first stage of our algorithm where we initialize the weight matrix W is given in Algorithm 1. To deal with a score note s spanning several time points, variable *currentTimePoint* scans through all time points $u \in U$ such that $\sigma(s) \leq u < \tau(s)$. Thus for each $(s, \textit{currentTimePoint})$ pair we update all d elements of the column of W corresponding to *currentTimePoint* as needed.

```
1  initialize all W entries to 0;
2  s ← first score note;
3  loop
4  |  currentTimePoint ← σ(s);
5  |  while currentTimePoint < τ(s) do
6  |  |  for j ← 0 to d − 1 do
7  |  |  |  if f(π(s), j) > W(j, currentTimePoint) then
8  |  |  |  |  W(j, currentTimePoint) ← f(π(s), j);
9  |  |  currentTimePoint ← next time point in U;
10 |  s ← next score note;
   end
```

Algorithm 1: Precomputing weight matrix according to f

Recall that W has size $O(dn)$. Each of these $O(dn)$ matrix positions will be updated at most l times, where l is the maximum polyphony of the score. This gives a total computation time for W of $O(dln)$. Keep in mind that l is often significantly smaller than d and can often be treated as a small constant.

The pseudocode for the second stage of our algorithm where we initialize our data structures for the main loop is given in Algorithm 2. Recall that the pointer corresponding to the start of pattern note p is p_σ, while the pointer corresponding to the end of p is p_τ. We store the initial horizontal translations based on these pointers in heap *NextEvent*.

```
1  p ← first pattern note;
2  loop
3  |  p_σ ← u_1;
4  |  p_τ ← u_1;
5  |  Min-Heap-Insert(NextEvent, p_σ − σ(p));
6  |  Min-Heap-Insert(NextEvent, p_τ − τ(p));
7  |  p ← next pattern note;
   end
```

Algorithm 2: Initializing pointers for candidate translations

Assigning values to the pointers and the translations take $O(1)$ time, while inserting into a heap takes $O(\log m)$ time. Therefore the second stage of our algorithm takes

$O(m \log m)$ time.

Finally we turn to the main and final stage of our algorithm. The pseudocode is given in Algorithm 3. Line 18 of the pseudocode refers to an *UpdateWeights* subroutine, and the pseudocode for that is given in Algorithm 4. First examining Algorithm 3, the variables *BestWeight* and *BestTranslation* store the optimum weight over all candidate translations seen so far, and the translation needed to achieve that weight respectively. We use *currentTrans* and *prevTrans* to store the horizontal translations corresponding to the current and previous events, respectively.

```
1  BestWeight ← 0;
2  BestTranslation ← (0, 0);
3  for π ← −(d − 1) to d − 1 do
4  |  for each pattern note p do
5  |  |  δ_π(p) ← 0;
6  |  w_π ← 0;
7  |  Δ_π ← 0;
8  currentTrans ← 0;
9  while NextEvent ≠ ∅ do
10 |  prevTrans ← currentTrans;
11 |  currentTrans ← Heap-Extract-Min(NextEvent);
12 |  p ← pattern note associated with currentTrans;
13 |  ptr ← pointer associated with currentTrans;
14 |  noteStart ← True;
15 |  if ptr = p_τ then
16 |  |  noteStart ← False;
17 |  for π ← −π(p) to d − 1 − π(p) do
18 |  |  UpdateWeights();
19 |  if next(ptr) exists then
20 |  |  ptr ← next(ptr);
21 |  |  trans ← ptr −σ(p);
22 |  |  if ptr = p_τ then
23 |  |  |  trans ← ptr −τ(p);
24 |  |  Min-Heap-Insert(NextEvent, trans);
25 return (BestWeight, BestTranslation);
```

Algorithm 3: Computing the optimum translation

In lines 1–8 of Algorithm 3, we provide initial values for all of the variables that we will be updating. The contents of the while loop at line 9 define the actions that our algorithm performs for each event. Lines 10–16 correspond to the updating the current translation

value by extracting the minimum value from heap *NextEvent*, as well as determining whether the current pointer corresponds to the start or end of pattern note p. For each possible transposition π, we call the *UpdateWeights* subroutine at line 18. Lines 19–24 moves the current pointer further along in U, as long as there are time points remaining, and inserts the new translation into heap *NextEvent*.

The initialization phase of Algorithm 3 (lines 1–8) takes $O(dm)$ time. The while loop at line 9 executes once for each event, and there are $O(nm)$ events. Lines 11 and 24 each take $O(\log m)$ time for the heap extraction and insertion operations, while all other lines inside the while loop run in constant time, except for the for loop at line 17. This for loop executes in $O(d)$ time. Therefore the complete running time for the final stage of our algorithm is $O(nm(dk + \log m))$, where subroutine *UpdateWeights* runs in $O(k)$ time. As we can see from Algorithm 4 below, subroutine *UpdateWeights* runs in constant time, and so the time complexity of our entire algorithm is $O(nm(d + \log m))$ as claimed.

```
1  w_π ← w_π + (currentTrans − prevTrans)Δ_π;
2  if w_π > BestWeight then
3  |  BestWeight ← w_π;
4  |  BestTranslation ← (currentTrans, π);
5  ν ← π(p) + π;
6  δ_π ← 0;
7  prevWeight ← 0;
8  nextWeight ← 0;
9  if prev(ptr) exists then
10 |  prevWeight ← W(ν, prev(ptr));
11 if next(ptr) exists then
12 |  nextWeight ← W(ν, ptr);
13 if noteStart then
14 |  δ_π ← prevWeight − nextWeight;
15 else δ_π ← nextWeight − prevWeight;
16 δ_π(p) ← δ_π(p) + δ_π;
17 Δ_π ← Δ_π + δ_π;
```

Algorithm 4: The contents of subroutine *UpdateWeights*

Examining the *UpdateWeights* subroutine, the weight at the current translation is updated in line 1, using the prior value of Δ_π. We track whether this is the best weight seen so far in lines 2–4. We then calculate the value for δ_π by defining *prevWeight* and *nextWeight* in lines 5–15 as described in Subsection 4.6.1, and update $\delta_\pi(p)$ and Δ_π appropriately in lines 16–17. Note that while we maintain m different $\delta_\pi(p)$ values for each transposition π, we only update one of these at each event, because each event is associated with a particular pattern note p.

4.7 Enhancements

Our algorithm as described above gives us a solid base to build upon, in terms of increasing its efficiency and its capabilities. We briefly describe some of the extensions that can be easily applied to our basic algorithm; potential extensions that we have not yet explored will be discussed in Chapter 7.

4.7.1 Efficiency

In the first stage of our algorithm, we saw that it requires $O(dln)$ time to compute all of the entries in weight matrix W. We assumed l was constant. For example, in violin music l is bounded by 4, while for piano music 10 is a natural bound for l. For symphonic music l is bounded by the number of notes that can be played simultaneously by the orchestra. As we see from this discussion, l can be a very large constant. There is a way to make the computation of W more efficient in terms of l.

We can reduce the time taken to calculate the weight values by computing W across rows, and using a heap to track the best weight for each entry in W. Since the polyphony of the score is l, each heap will have at most l values at any time, and insertion and deletion of heap values will take $O(\log l)$ time. Therefore we can reduce the running time of the first stage of our algorithm to $O(dn \log l)$ by computing entries in W across rows.

As noted in Section 4.1, if the $O(dn)$ space required by our weight matrix W is considered too prohibitive, we can significantly reduce the space complexity at the cost of additional time complexity by calculating weights during the final stage of our algorithm. Here we would eliminate the weight matrix W, and compute weight values as we need them during subroutine *UpdateWeights*. In subroutine *UpdateWeights*, when we have the weight matrix W, we used $W(\nu, prev(ptr))$ and $W(\nu, ptr)$ to compute the bookkeeping values used to determine how the weight will change from this event to a future event.

We will calculate these two values during each iteration of *UpdateWeights* by using a smaller set of precomputed stored data. We are already given ν and ptr. For each time point, we can precompute a list of score note pitches that are active at that moment in time. That is, for each time point u, we store $\pi(s)$ for each score note s such that $\sigma(s) \leq u < \tau(s)$. This would replace the first stage of our basic algorithm, and would take $O(nl)$ time and space.

Given these precomputed lists of pitches for each time point, we can simply compare $f(\pi(s), \nu)$ values for all active score notes s at the time point corresponding to ptr in order to compute $W(\nu, ptr)$ in *UpdateWeights*. We also do the same for the $W(\nu, prev(ptr))$ value. This would require $O(l)$ additional work in *UpdateWeights*, since each precomputed list has size at most l. Therefore the trade-off for reducing the space complexity from $O(dn + m)$ to $O(nl + m)$ is an increase in the running time from $O(nm(d + \log m))$ to $O(nm(dl + \log m))$.

4.7.2 Finding the Best k Matches

Our basic algorithm only returns the best match. We can easily make this more useful by returning the k best matches; the obvious approach to this would be to keep track of the k best matches at each of the $O(nm)$ events in a heap, and at each event, compare the weight of the current match to the worst match stored in the heap, and update appropriately. This approach would increase the running time of our basic algorithm from $O(nm(d + \log m))$ to $O(nm(d \log k + \log m))$, and would change the space complexity of our algorithm by adding $O(k)$ space.

A different approach can accomplish this without increasing our running time. We maintain an array of size $2k$ to store (*weight*, *translation*) pairs as follows: for the first k events, we store the resulting (*weight*, *translation*) pairs into the first k array entries, and keep track of the worst weight seen over the first k events. This can be done in constant time in *UpdateWeights*.

For each successive event, we examine the weight of the current translation. If it is less than the worst weight we have stored in the array, we discard the current translation because it cannot have one of the best k weights. If the weight of the current translation is greater than the worst weight we have stored in the array, we add the current (*weight*, *translation*) pair to our array if it is not yet full.

Once we have $2k$ elements stored in our array, we apply the linear time median finding algorithm to the array to find the array entry with the k^{th} best weight. Once we have done so, the linear time median finding algorithm has partitioned the array so that the first half of the array contains the best k matches seen so far, while the second half of the array contains matches of lower weight. We then discard the k array entries in the second half of the array, and repeat the process until all events have been examined, and we are left with the k best matches overall.

The linear time median finding algorithm takes $O(k)$ time, and our array requires $O(k)$ space. Note that the (*weight*, *translation*) pair associated with a particular event is either stored and kept in the array (if it is one of the k best matches), discarded immediately, or discarded during an application of the linear time median finding algorithm. In the latter case, k elements are discarded with the median finding algorithm taking $O(k)$ time. Therefore over the course of the entire algorithm, $O(1)$ work is performed for each of the $O(nm)$ events in order to keep a good match, or discard a bad match, and the time complexity of our algorithm does not increase.

4.7.3 Matching Note Starts

As we shall see in Chapter 6, it will be desirable to augment our interval-based weight model to give additional weight to translated pattern notes that start at the same time as score notes. This is a good example of a useful weight model that is not purely based on

pitch. It is easy to keep track of whether the current event matches the start of a pattern note to the start of a score note; we can maintain additional book-keeping information to determine this in constant time. If the current event matches the start of a pattern note to the start of a score note, the challenge is to determine whether the pattern note matches best to that particular score note.

Suppose that a pattern note p matches best against score note s_1 by contributing weight w_1, and p does not match well against score note s_2—such a match would contribute weight $w_2 < w_1$. See Figure 4.11. Note that p does not start at the same time that s_1 starts, while p does start at the same time as score note s_2. If we now add some sort of additional weight w_{start} for matching note starts, the weight contributed by matching p to s_2 becomes $w_2 + w_{start}$. In this case, we would like to compare $w_2 + w_{start}$ to w_1 and choose the better weight.

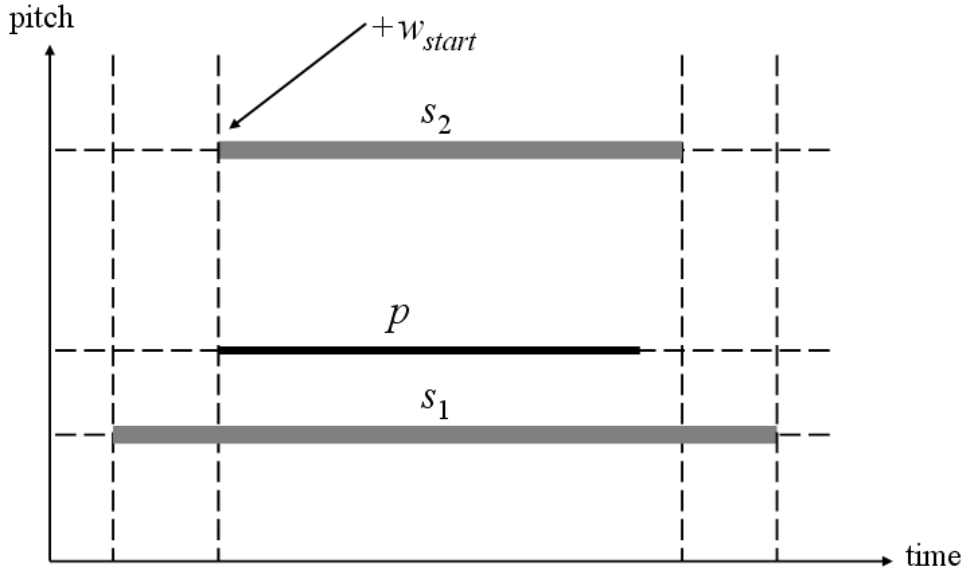


Figure 4.11: Ordinarily p would match well to s_1, but if we have additional weight for matching note starts, p could have a better match to s_2.

We can give additional weight to matching note starts by precomputing a second weight matrix, W_{start}. Recall that $W(\pi, u)$ is the weight to be applied to the portion of a pattern note lying between time point u and the next time point u'. We define $W_{start}(\pi, u)$ to be the weight applied to a pattern note that starts at time point u. If no score notes start at time point u, then $W_{start}(\pi, u) = W(\pi, u)$ for all π. However, if one or more score notes start at time point u, then we will have $W_{start}(\pi, u) \geq W(\pi, u)$ for all u. We can precompute W_{start} by using a slightly modified version of the first stage of our algorithm.

Given this second weight matrix, we can take additional weights for matching note starts into account by making some modifications to the *UpdateWeights* subroutine. See Algorithm 5.

```
1  if currentTrans > prevTrans then startBonus ← 0;
2  ν ← π(p) + π;
3  if noteStart then
4  |   noteBonus ← W_start(ν, ptr) − W(ν, ptr);
5  |   startBonus ← startBonus + noteBonus;
6  w_π ← w_π+ (currentTrans − prevTrans)Δ_π + startBonus;
7  if w_π > BestWeight then
8  |   BestWeight ← w_π;
9  |   BestTranslation ← (currentTrans, π);
10 δ_π ← 0;
11 prevWeight ← 0;
12 nextWeight ← 0;
13 if prev(ptr) exists then
14 |   prevWeight ← W(ν, prev(ptr));
15 if next(ptr) exists then
16 |   nextWeight ← W(ν, ptr);
17 if noteStart then
18 |   δ_π ← prevWeight − nextWeight;
19 else δ_π ← nextWeight − prevWeight;
20 δ_π(p) ← δ_π(p) + δ_π;
21 Δ_π ← Δ_π + δ_π;
```

Algorithm 5: Modified version of subroutine *UpdateWeights* that handles additional weight for matching note starts

If we compare Algorithm 5 to Algorithm 4, we see that line 1 is new, line 2 has simply been moved earlier in the subroutine without affecting anything, lines 3–5 are new, and line 6 is modified. The rest of the subroutine proceeds normally—note that all of our bookkeeping values, which are based on the weights in W only, are maintained in exactly the same way. We therefore calculate any additional weight due to matching note starts by tracking the bonus weight separately, and applying any bonus weight to the weight of the current translation before comparing w_π to *BestWeight*.

Line 1 resets the bonus weight stored to zero if we actually perform a non-zero horizontal translation to get from the previous event to the current event. This makes sense because we have not yet examined the first event at the current translation. Lines 3–5 allow us to determine any additional weight we should add to *startBonus*, if the current event involves the start of a pattern note. We can easily recover the appropriate bonus

weight to be applied, if any, by taking the difference between corresponding entries in the two matrices as seen in line 4.

When it is time to update w_π in line 6, we add any bonus weight accumulated in *startBonus* to w_π before comparing w_π to *BestWeight*. If there are multiple events at this same translation that match notes starts, the last event at this translation will yield the correct weight for the match. This is because a weight bonus for matching note starts operates on a note-by-note basis. If we do not need to take matching note starts into account, the first event at a translation with multiple events yields the correct weight. If we do take matching note starts into account, we have to accumulate the bonus contributed by each pattern note involved, which is the reason why we recover the correct weight for this match at the last event of the translation.

The modifications to *UpdateWeights* still allow the subroutine to run in $O(1)$ time. Therefore we are able to provide a weight bonus for matching note starts without increasing the time complexity of our algorithm, while using another matrix of size $O(dn)$.

Chapter 5

Barriers to Faster Music Pattern Matching

As explained in the previous chapter, our algorithm runs in $O(nm(d+\log m))$ time. While previous music pattern matching algorithms do not have running times that depend on d, they sacrifice the power and flexibility of general weight models that our algorithm can use to define musical similarity in different ways. It would be desirable to reduce our running time's dependence on d, and we briefly investigate some related work towards this direction in Section 5.2. We first turn to the nm factor.

Many music pattern matching algorithms have quadratic running times, as seen in Section 4.2. In fact, only the naive approach to monophonic music based on basic string matching takes linear time, via the Knuth-Morris-Pratt algorithm [26]. If we restrict ourselves to algorithms for polyphonic music, no subquadratic algorithm has yet been developed. This dependence on nm in the geometric setting is not too surprising, given the need to take information about both pitch and duration into account for all notes in the pattern and score.

In a music information retrieval system using a large database of music, having each query run in $O(nm)$ time could quickly become impractical. Some approaches use indexing techniques from the databases field in order to quickly find common musical fragments, as the n-gram approach described in Subsection 3.1.3 does. Another approach to reducing the amount of time used to query a musical database could involve finding a subquadratic algorithm for the music pattern matching problem, which would be a very significant advance.

We will show that finding such subquadratic algorithms for the music pattern matching problem in the geometric setting will be a major challenge. In particular, we will show that a special case of the music pattern matching problem corresponds to a problem from computational geometry called "Segments Containing Points", or SCP. The SCP problem

is at least as hard as a basic problem called 3SUM, and it is conjectured that there is no algorithm to solve 3SUM in subquadratic time. Therefore, a subquadratic algorithm to solve the music pattern matching problem in the geometric setting would have major implications in algorithm design.

5.1 3SUM and SCP

The 3SUM and SCP problems are defined as follows:

3SUM: Given a set S of n integers, does S contain integers a, b, and c such that $a + b + c = 0$?

For example, a "yes" instance of the 3SUM problem could be $S = \{-11, -8, 0, 3, 5, 11\}$ for $(a, b, c) = (-11, 0, 11)$. A "no" instance of the 3SUM problem could be $S = \{-11, -3, 4, 8\}$—there is no way to choose three elements from S that sum up to zero.

SCP: Given a set P of n real numbers and a set Q of m pairwise-disjoint intervals of real numbers, does there exist a translation u such that $P + u \subseteq Q$?

A "yes" instance of SCP could have $P = \{-5, -1, 1, 4, 10, 12\}$ and $Q = \{[-17, -12], [-11, -8], [-7, -6], [-5, -3], [-1, 5]\}$, since $P + u \subseteq Q$ for $u = -8$. A "no" instance of SCP could have $P = \{-7, -2, 6\}$ and $Q = \{[-13, -10], [-6, -4], [4, 5], [9, 16]\}$, since no real number u exists such that $P + u \subseteq Q$. See Figure 5.1.

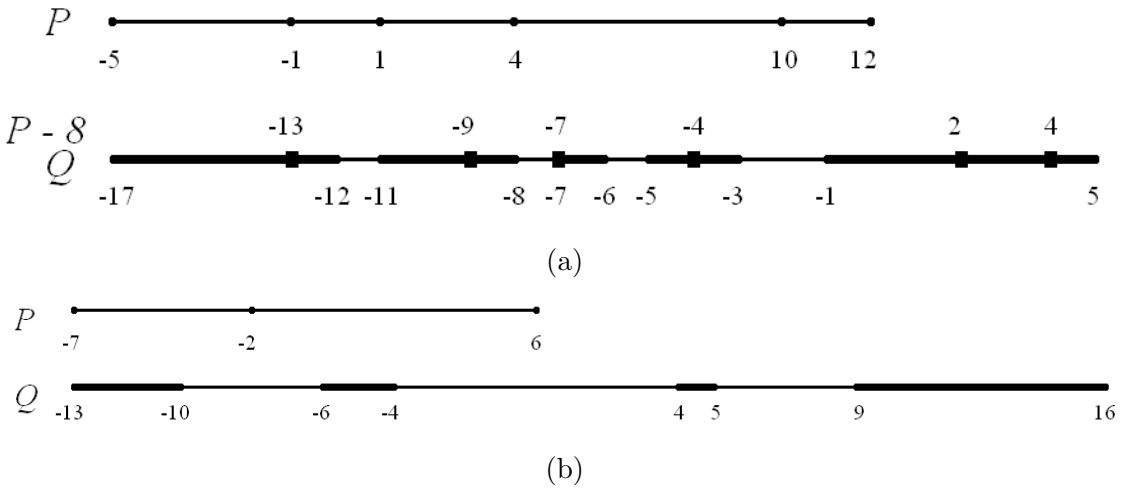


Figure 5.1: (a) A "yes" instance of the SCP problem—a suitable translation is given by $P - 8$, shown by squares in intervals of Q; (b) a "no" instance of the SCP problem.

Note that SCP is a special case of exact matching in the music pattern matching problem. In particular, SCP can represent an instance of the music pattern matching problem where all of the score and the pattern notes have the same pitch, with the pattern notes having extremely short durations. Therefore, it follows that any algorithm to solve geometric music pattern matching can be applied to solve the SCP problem. The music pattern matching problem in the geometric setting is at least as difficult as SCP.

The SCP problem, in turn, is one of many problems in computational geometry that are at least as difficult as 3SUM (see Theorem 2 below.) We say that SCP is "3SUM hard". Gajentaan and Overmars [19] present a number of different 3SUM hard problems, and note that the most efficient known algorithm to solve the 3SUM problem runs in quadratic time, $O(n^2)$. This is not a proved lower bound, but the continuing difficulty of finding a subquadratic algorithm to solve 3SUM has led to the conjecture that no subquadratic algorithm to solve 3SUM exists.

Barequet and Har-Peled show that SCP is at least as hard as 3SUM by proving that a subquadratic algorithm for SCP would imply a subquadratic algorithm for 3SUM. We include the proof in order to provide some insight on how the music pattern matching problem relates to the 3SUM problem.

We first prove that 3SUM' is equivalent to 3SUM, where 3SUM' is defined as follows:

3SUM': Given three sets A, B, and C of n integers each, does there exist $a \in A$, $b \in B$, and $c \in C$ such that $a + b = c$?

Theorem 1 *3SUM' is equivalent to 3SUM. [19]*

Proof: We first show that 3SUM' is at least as hard as 3SUM. Given an instance S of 3SUM, we construct an equivalent instance (A, B, C) of 3SUM' as follows. Set $A = S$, $B = S$, and $C = -S$. Then if there exist $a \in A$, $b \in B$, and $c \in C$ with $a+b = c$ we must have $a, b, (-c) \in S$ such that $a + b + (-c) = 0$.

In order to prove the other direction, suppose that we are given an instance (A, B, C) of the 3SUM' problem. We will create an equivalent instance S of 3SUM as follows: first, without loss of generality we assume that all elements in A, B, and C are positive. (If they are not, we can take some large number k such that the sets $A + k$, $B + k$, and $C + 2k$ is an equivalent 3SUM' instance such that all elements are positive.)

Define $m = 2 \times \max(A, B, C)$. We construct S as follows: for every $a \in A$, add $a' = a + m$ to S; for every $b \in B$, add b to S; for every $c \in C$, add $c' = -c - m$ to S. If $a + b = c$, then $a' + b' + c' = a + m + b - c - m = a + b - c = 0$.

Now we show that if three elements of S add up to 0, they must have originally been derived from sets A, B, and C. From the construction of S above, we have $a' \in (m, 1.5m]$, $b' \in (0, 0.5m]$, and $c' \in [-1.5m, -m)$ for all $a \in A$, $b \in B$, and $c \in C$.

Suppose we have $x, y, z \in S$ with $x + y + z = 0$. At most one of x, y, and z can originally come from A because otherwise we would have $x + y + z > 2m - 1.5m = 0.5m$. Similarly, at most one of x, y, and z can originally come from C because otherwise we would have $x + y + z < -2m + 1.5m = -0.5m$. Since all elements in S that originally come from A and B are positive, exactly one of x, y, and z must originally come from C; without loss of generality, suppose this element is z.

Now suppose x and y both originally come from B. Then $x + y \leq m$, while $z < -m$, and so $x + y + z < 0$. Therefore one of x or y must come from A, and if $x, y, z \in S$ with $x + y + z = 0$, x, y, and z must originally come from different sets A, B, and C.

Therefore 3SUM is equivalent to 3SUM'. □

Theorem 2 *SCP is 3SUM hard. [3]*

Proof: We prove that SCP is 3SUM hard by showing that it is 3SUM' hard. First it is useful to define an intermediate problem. The Equal Distance (ED) problem will be used to relate 3SUM' to SCP.

ED: Given two sets P and Q of n and m real numbers, respectively, does there exist $p_1, p_2 \in P$ and $q_1, q_2 \in Q$ such that $p_1 - p_2 = q_1 - q_2$?

We first prove that ED is 3SUM' hard by reducing 3SUM' to ED. We start with (A, B, C), an instance of 3SUM' with A, B, and C each having n integers. Using affine transformations, we can limit all elements of $A \cup B \cup C$ to the open interval (0, 1) without changing whether this is a "yes" instance or a "no" instance.

For example, suppose we have an instance (A, B, C) of 3SUM' with $A = \{4, 9, 17.5, 29\}$, $B = \{-3, -1, 8.3\}$, and $C = \{2.8, 6, 28.5\}$. After adding 4 to A and B, and 8 to C, we obtain $A = \{8, 13, 21.5, 33\}$, $B = \{1, 3, 12.3\}$, and $C = \{6.8, 14, 36.5\}$. After dividing all elements by 40, we finally get $A = \{0.2, 0.325, 0.5375, 0.825\}$, $B = \{0.025, 0.075, 0.3075\}$, and $C = \{0.17, 0.35, 0.9125\}$ to limit all elements of $A \cup B \cup C$ to the open interval (0, 1).

The corresponding ED instance is constructed as follows: we create a set P of $2n$ real numbers by mapping each $c_i \in C$ to a pair of numbers in P with difference $3 - c_i$. Formally,

$$P = \{100i, 100i + 3 - c_i | c_i \in C, i = 1, \ldots, n\}$$

We also create a set Q of $2n$ real numbers defined as follows:

$$Q = A \cup \{3 - b | b \in B\}$$

To show that (P, Q) is an instance of the ED problem that corresponds to our 3SUM' instance, we first assume that there exists a solution to the 3SUM' instance and prove that there must exist a solution to the ED instance. A solution to the 3SUM' instance consists of $a \in A$, $b \in B$, and $c_i \in C$ such that $a + b = c_i$. Then choose $p_1 = 100i + 3 - c_i$ and $p_2 = 100i$, $p_1, p_2 \in P$. $p_1 - p_2 = 3 - c_i = 3 - (a + b) = (3 - b) - a$, and by the definition of Q, $q_1 = 3 - b$, $q_2 = a \in Q$ to satisfy the ED instance.

Now assume that there exists a solution to the ED instance; we must prove that a solution exists for the corresponding 3SUM' instance. Therefore we must have $p_1, p_2 \in P$ and $q_1, q_2 \in Q$ such that $p_1 - p_2 = q_1 - q_2$. Due to the way that P and Q were constructed, note first that $q_1 - q_2 < 3$ since $A \cup B \subseteq (0, 1)$. Hence we must have that $p_1 - p_2 < 3$, with p_1 and p_2 corresponding to the same $c_i \in C$—if p_1 and p_2 corresponded to different elements of C, then $p_1 - p_2 > 97$. Furthermore, $p_1 - p_2 > 2$ because $C \subseteq (0, 1)$. This ensures that $q_1 - q_2 > 2$ as well, and therefore q_1 and q_2 cannot both be in $A \subseteq (0, 1)$ or $\{3 - b | b \in B\}, B \subseteq (0, 1)$. Therefore we must have $100i, 100i + 3 - c_i \in P$ and $a, 3 - b \in Q$ where $a \in A$, $b \in B$, and $c_i \in C$ such that $(100i + 3 - c_i) - (100i) = (3 - b) - a$. Therefore $-c_i = -b - a$, and so $a + b = c_i$, a solution to the corresponding 3SUM' instance. Therefore ED is 3SUM' hard.

We now prove that SCP is 3SUM' hard by reducing 3SUM' to SCP, making use of the ED instance corresponding to the 3SUM' instance in between. We construct an instance (P, Q') of SCP by reusing P as defined in the ED instance, and defining Q' as follows:

$$Q' = [-100(n - 1), -94] \cup Q \cup [100, 100(n - 1) + 6]$$

In this case, we interpret each $q \in Q$ as the interval $[q, q] \in Q'$.

To show that (P, Q') is an instance of the SCP problem that corresponds to our 3SUM' instance, we first assume that there exists a solution to the 3SUM' instance and prove that there must exist a solution to the SCP instance. There exists $a \in A$, $b \in B$, and $c_i \in C$ such that $a + b = c_i$. In the corresponding ED instance, we must have $p_1, p_2 \in P$ corresponding to c_i and $q_1 = a$, $q_2 = 3 - b \in Q$. Therefore in the SCP instance, the translation of p_1 and p_2 is covered by the intervals $[q_1, q_1]$ and $[q_2, q_2] \in Q'$. Any $p \in P$, $p \neq p_1$ or p_2 must have the property that $|p - p_1| > 97$ and $|p - p_2| > 97$. Therefore any $p \in P$ that is not p_1 or p_2 must lie in the interval $[-100(n - 1), -94]$ or the interval $[100, 100(n-1)+6]$, due to the definitions of P and Q'. Hence there exists some translation u such that $P + u \subseteq Q'$, and the corresponding SCP instance is satisfied.

Now assume that there exists a solution to the SCP instance; we must prove that a solution exists for the corresponding 3SUM' instance. We have $P + u \subseteq Q'$ for some real number u. Note that the length of each of the two new intervals in Q' is $100n - 194$, and the difference between the lowest and greatest values in P is at least $100n - 98$. The gap between the two new intervals has length 194, so we cannot have $P + u \subseteq (Q' - Q)$. Therefore, at least one of the original numbers $q \in Q$ must coincide with some $p + u$,

$p \in P$. This p corresponds to some $c_i \in C$. Due to the way that Q' is defined, the other $p' \in P$ that corresponds to c_i must have the property that $p' + u = q'$ for some $q' \in Q$, since the SCP instance is satisfied. Therefore we have $p + u = q$ and $p' + u = q'$, and so $p - p' = q - q'$, satisfying the corresponding ED instance. This implies that there exists $a \in A$ and $b \in B$ such that $a + b = c_i$, and therefore a solution to the corresponding 3SUM' instance exists.

Therefore SCP is 3SUM' hard, and therefore is 3SUM hard as well.

□

Barequet and Har-Peled go on to show that some other geometric problems are also 3SUM hard. In particular, they show that certain polygon containment problems are at least as difficult as SCP—while the problem of determining whether it is possible to translate a convex polygon P such that it is contained within Q can be solved in linear time, variants of this problem involving non-convex polygons or rotation are at least as hard as SCP, and therefore are unlikely to have subquadratic algorithms. Similarly, the problem of determining the minimum Hausdorff distance between two sets of line segments in the plane is also 3SUM hard.

5.2 Implications for Music Pattern Matching

Therefore we see that finding a subquadratic algorithm to solve the music pattern matching problem in the geometric setting would imply the existence of subquadratic algorithms for SCP, 3SUM, and other related problems. Because 3SUM is the base problem, finding a subquadratic algorithm for 3SUM would be easier, or at least no more difficult than attempting to find a subquadratic algorithm for the music pattern matching problem. Since attempts to find a subquadratic algorithm for 3SUM have been unsuccessful, it is not surprising that similar attempts have failed for the music pattern matching problem. Therefore attempting to reduce our running time's dependency on nm is unlikely to be useful.

In practice, m is usually significantly smaller than n. For example, a typical piece of music may last for five minutes, with corresponding queries using patterns with durations that are 50 to 100 times shorter than the entire piece. As we shall see in the next chapter, typical values of m are less than twenty. Since this is the case, the dependence on the pitch set $d = 128$ is a greater factor than the dependence on m. Therefore, in order to make our algorithm more efficient, attempting to reduce the dependence on d will likely yield better results than attempting to reduce the dependence on m.

Finally, we mention a related positive result. Efrat, Indyk, and Venkatasubramanian [14] present an approach to solve the geometric pattern matching problem for sets of axis-aligned line segments using a *coverage measure* based on the l_∞ norm, along with

a parameter $\epsilon > 0$ to handle approximate matching. See Figure 5.2. In this case, we have $\epsilon = 1$, and outline the areas where a line segment of A can cover a point of a line segment in B by a light grey rectangle in Figure 5.2(b). The coverage measure in this case is $Cov_\epsilon(A, t + B) = 5$, by summing the length of B's line segments that lie within the grey rectangles.

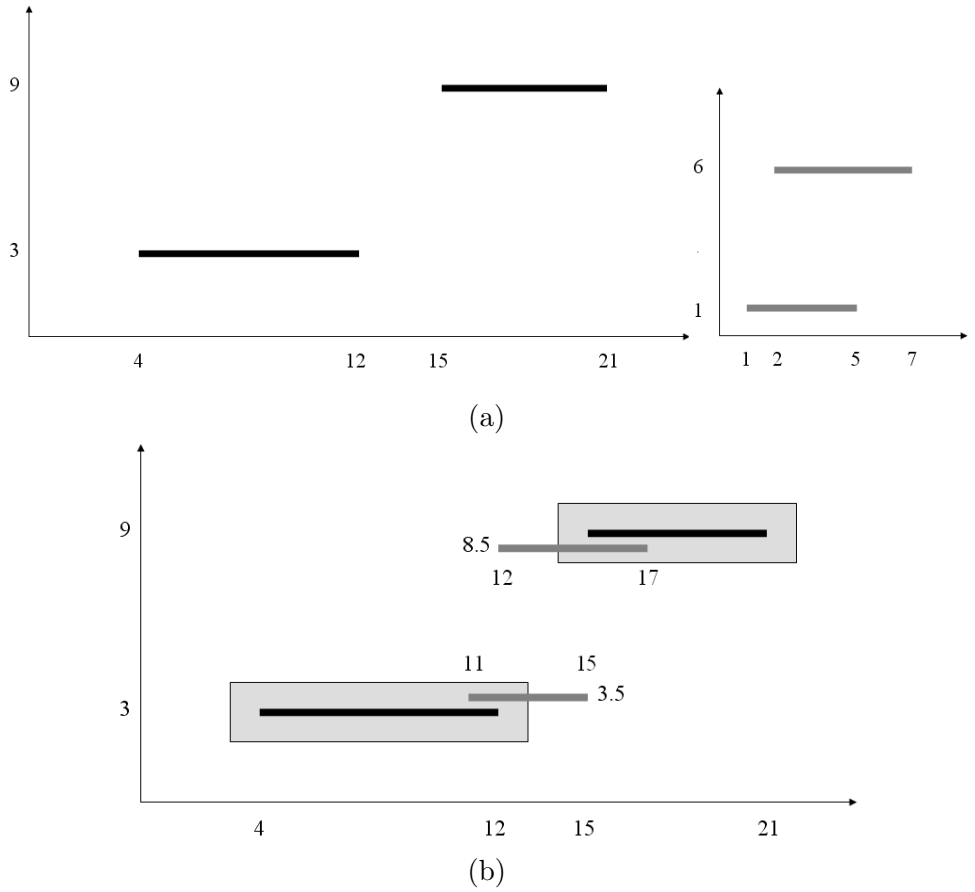


Figure 5.2: (a) Two sets of horizontal line segments A (black) and B (grey); (b) Calculating the coverage measure $Cov_\epsilon(A, t+B)$ between A and B under translation $t = (10, 2.5)$ and with $\epsilon = 1$.

For two sets of n horizontal line segments, the authors find the translation of one set of segments relative to the other that maximizes the coverage measure between the two sets in $O(n^2 \log^2 n)$ time. This is notable because it does not depend on a discrete pitch set; the vertical coordinates of the horizontal line segments can be any real numbers. This approach may shed insight towards how to reduce our algorithm's dependence on the discrete pitch set, although the coverage measure presented is not well-suited to applying musically sensible approximate matching.

Chapter 6

Experiments

In this chapter we report on experiments that demonstrate the capabilities of our algorithm. These experiments do not provide any comparison of computation time, as our implementation was done in the high-level language Mathematica. While it will be instructive in the future to compare the computation time of an efficient implementation of our algorithm against other algorithms for music pattern matching, our current goal is to highlight the musically sensible matches our algorithm can recognize when using the interval-based weight model.

We divide our experiments into two categories: those searching for matches of a monophonic pattern into a polyphonic score, and those searching for matches of a polyphonic pattern into a polyphonic score. In each category we examine two scores, and then examine matches of several patterns into the scores. Most of the patterns we use have an exact occurrence in the score, so that approximate matches indicate musically sensible variations of that pattern elsewhere in the score. Some patterns do not have an exact occurrence in the score, but are adapted from similar works by the same composer, in order to see how similar themes are between musical works.

Our scores are taken from Baroque and Classical music from the 18^{th} century, from well-known composers such as J. S. Bach, Haydn, and Mozart. We also focus primarily on piano music, due to its inherent polyphony as well as common availability. Although it would be interesting to experiment on other kinds of music, we consider this a good starting point because these works are complex enough so that a casual observer would find it difficult to find all of the musically sensible matches that our algorithm finds, at a glance. Also, such music also exhibits some degree of regularity, with variations on monophonic and polyphonic themes often occurring many times in a single score. Therefore this setting provides a good challenge for our algorithm, with the scores still admitting a few patterns that produce a good number of musically sensible matches.

The values of n, m, and d used in our experiments are given in Table 6.1. In each case we obtain d by examining the pitch range between the highest score note pitch and the lower score note pitch, in MIDI pitches.

Score	Pattern	n	m	High Pitch	Low Pitch	d
Bach Invention	all 3 patterns	471	8	84	36	49
Mozart Sonata	1st pattern	1755	11	88	35	54
Mozart Sonata	2nd pattern	1755	9	88	35	54
Bach Fugue	only pattern	749	12	84	36	49
Haydn String Quartet	only pattern	1808	28	91	38	54

Table 6.1: Values of n, m, and d used in our experiments

We obtain sheet music images and corresponding MIDI files from resources in the public domain, such as the Mutopia Project [43]. In most cases these images and MIDI files were generated by contributors to the Mutopia Project using GNU LilyPond [38], an "automated engraving system." MIDI files generated using LilyPond contain precise information about note start and end times that exactly match the musical notation, which allows us to avoid the irregularities that occur in MIDI files produced through human performances. We do examine one example of a human-produced MIDI file in our experiments, in order to see how our algorithm can handle more adverse conditions.

The interval-based weights used in our experiments are given in Table 6.2.

Interval apart (in semitones)	Weight
perfect unison (0)	1
minor 2nd (1)	0.9
major 2nd (2)	0.6
minor 3rd (3)	0.4
major 3rd (4)	0.4
perfect 4th (5)	0.2
perfect 5th (7)	0.6
minor 6th (8)	0.3
major 6th (9)	0.3
major 7th (11)	0.7
perfect octave (12)	0.8
minor 9th (13)	0.7
all other intervals	0

Table 6.2: The weighting scheme used for our experiments

The interval-based edit distance weights put forth by Mongeau and Sankoff [37] are slightly similar to our weight values. Consecutive MIDI pitches are one semitone apart, and so we model the different pitches in an octave with 12 consecutive semitones, instead of scale degrees. Therefore we give very good weights to pitches that differ by a single semitone, since a pattern that starts at a different scale degree often differs by a single semitone for several notes. We also assign a very good weight to notes that are an octave apart, because they are the "same" note in a musical sense, with the higher note having exactly twice the frequency of the lower note.

For similar reasons as presented above, we also give good weights to the semitones on either side of an octave. Intervals beyond 13 semitones are assigned zero weight, to delimit a sensible vertical range for the pattern to match into the score. The other intervals presented in the table have weight corresponding roughly to how common these intervals occur in a harmonic sense; that is, a pattern note being replaced by a note that is a perfect 5^{th} higher should result in a better match than that pattern note being replaced by a note that is a perfect 4^{th} higher.

These particular values do yield meaningful results in our experiments, but the musical sensibility of other weight values can certainly be argued. In particular, note that if a pattern note G matches against a score note C, the interval between these two notes can be either a perfect 4^{th} or a perfect 5^{th}, depending on the score note changing by an octave. While this is consistent with the methodology of Mongeau and Sankoff, we note that this behaviour can be undesirable, and arises from a limitation of comparing pairs of notes to each other. A richer approach could potentially take the entire harmonic context at a particular time into account (for example, identify a specific chord), and we discuss this further in Chapter 7.

It would be interesting to apply learning techniques to a set of training weights in order to learn a "good" set of interval-based weights. This approach would expand our pattern matching problem to a more general method of music information retrieval.

We present the weight of a match of the pattern into the score as a percentage, with an exact match receiving a weight of 100%. From the weight values above, it is easy to see that a match of the pattern into the score where each note is a minor 2^{nd} apart would achieve a weight of 90%, while another match of the pattern that is exactly an octave apart from the score would yield a weight of 80%. From this we might deduce that "good" weights are in the 80% to 90% range, while "very good" weights are in the 90% to 100% range. This is not always the case, however, as the "threshold" value that separates good matches from bad matches (according to our weight model) depends heavily on characteristics of the pattern—which notes are in the pattern, how long the pattern notes are, and so on.

Therefore we are not really interested in the absolute values returned for the weight of each match; rather, we are concerned with the relative values, which show us how the quality of one match compares to another. The relative values can also indicate

how successful our algorithm is at separating good matches from bad matches. We will examine the circumstances under which a false positive occurs (when a bad match receives a good weight), and also look at cases where our algorithm may assign a poor weight to a musically sensible match.

We determine where the threshold should be by examining the matches that our algorithm finds, and applying our musical knowledge to check how often noticeable musical differences occur when comparing the matches to the corresponding notes in the score. In some cases we choose patterns consistent with those explored by other experts in previous approaches. While this can be rather subjective, there is no purely objective way to determine exactly how similar two musical fragments are, as has been previously discussed. This method does make a good deal of sense from a music theoretic point of view, however, and our experiments will show that our method can find more musically sensible matches than methods that rely purely on geometric properties.

6.1 Results

6.1.1 Monophonic Patterns

We first present some experiments that match monophonic patterns into polyphonic scores. We search for approximate occurrences of a melodic fragment in a Bach piano invention, and also examine a melodic theme from a Mozart piano sonata adapted from Selfridge-Field [44]. For each of these two scores, we use patterns taken directly from the scores, as well as patterns from other sources in order to demonstrate how our algorithm works.

Our first polyphonic score will be the Two-Part Invention, Number 1, BWV 772 by J. S. Bach. The first two bars of this score are shown in Figure 6.1. This is a short piano piece only 22 bars in length, and is often played in less than a minute. All of Bach's music exhibits some degree of regularity; melodic themes featured in his two-part inventions tend to occur repeatedly in each of the two parts (which are played by the left and right hands of the pianist.)

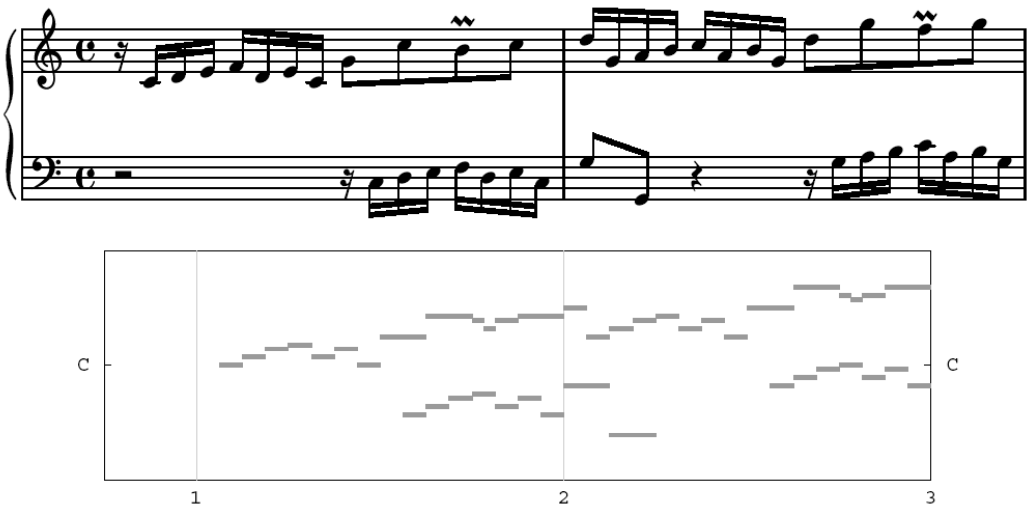


Figure 6.1: The first two bars of J. S. Bach's Invention 1, in common musical notation, and in the corresponding line segment representation.

The first pattern that we use is based mostly on the beginning of the piece, but in fact has only one exact occurrence in the entire score, at bar 19. See Figure 6.2. The pattern differs from the first eight notes of the piece only in the last note, which has a higher pitch at the start of the piece than it has in bar 19. Orpen and Huron [41] also discuss finding occurrences of the first eight notes of the Bach Invention elsewhere in the piece, so there is some musical basis behind our choice of pattern.

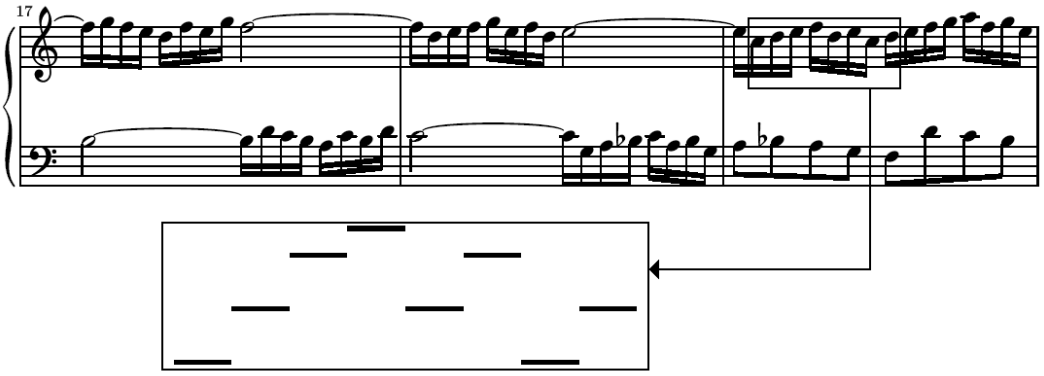


Figure 6.2: Our first pattern from bar 19 of Bach's Invention.

We search for the best 20 matches of our pattern into the score, according to our interval-based weight model. The results are shown in Figure 6.3.

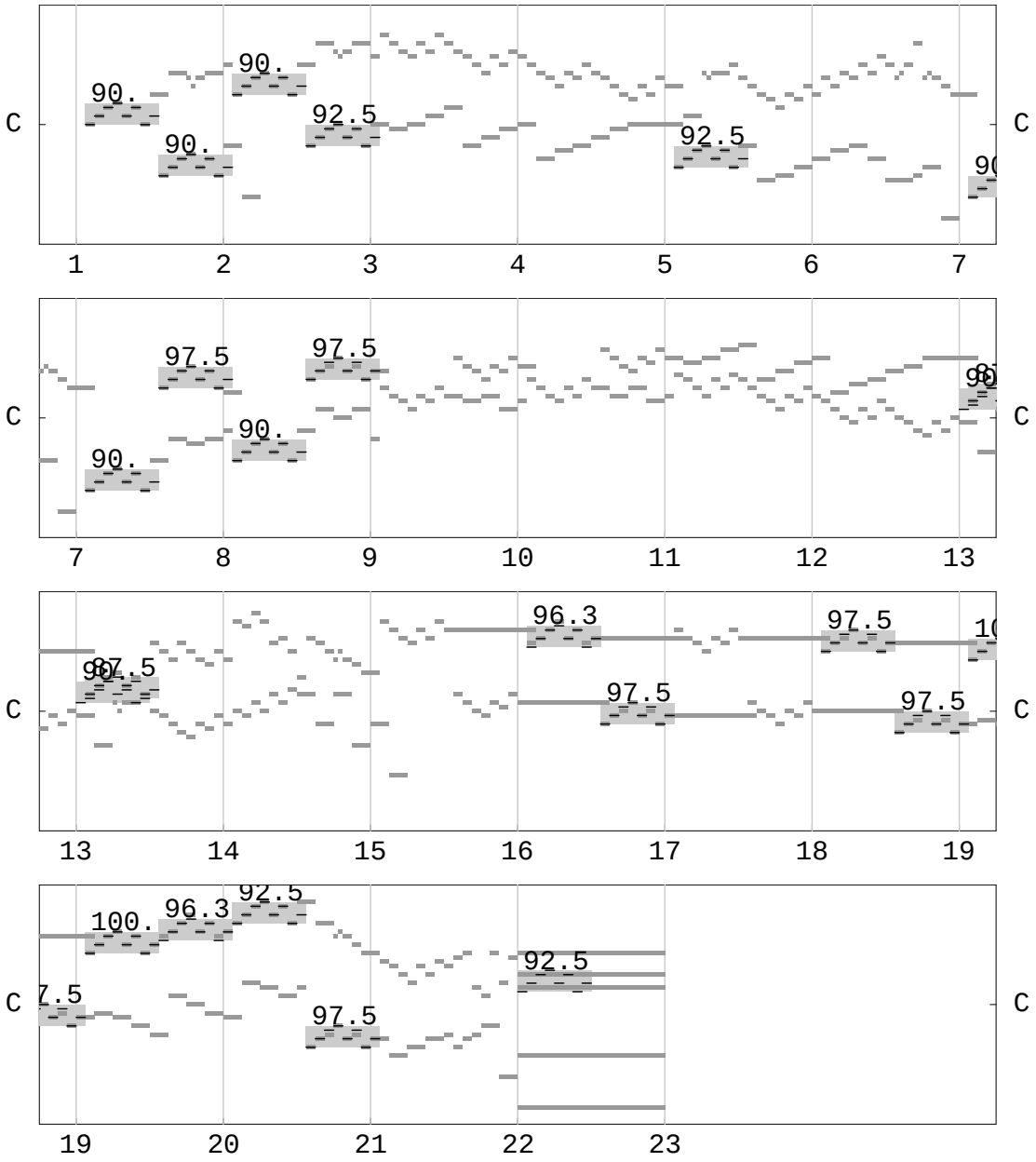


Figure 6.3: The entire line segment representation of the Bach Invention. Matches of our first pattern are indicated with black line segments and grey rectangles, and the quality of the matches are indicated with percentage values.

We discover that 19 matches would have been a better cut off, since the 20^{th} match (found in bar 13 with a weight of 87.5%) is not musically sensible. The poor match in

bar 13 "piggybacks" on the better 90% match found in the same bar, which is a good indication that the threshold between good matches and bad matches (according to our weight model) occurs at that point. There are 18 musically sensible matches in the piece, and our algorithm recognizes these and assigns corresponding weights between 90% and 100%. One extraneous match is found in the top 19 matches; this is the match in bar 22, with a weight of 92.5%.

This false positive is the result of two factors. First, our pattern has only four distinct pitches, and because pitches that are only a semitone apart receive very good weights under our interval-based weight model, our algorithm considers matching the pattern to the two notes of a minor 3$^{\text{rd}}$ of the final chord in the score to be a good match. Secondly, our interval-based weight model does not penalize translated pattern notes that do not start at the same time that score notes start. Only the first note of the translated pattern starts at the same time as a score note in that false positive; the remaining seven notes in the translated pattern do not.

If we assign a penalty to pattern notes that do not start at the same time that score notes start, we will be able to better handle the distinction between a chord (in which a few notes are being played for a long period of time) and a group of short notes being played in sequence (for example, our pattern.) For example, applying a penalty of 50% to the weight of pattern notes that do not start at the same time as score notes assigns the false positive in bar 22 a much more sensible weight value of 51.9%, while the other matches keep their high weights. Even a smaller penalty of 10% would correctly separate the false positive from the good matches.

For our second pattern, we simply invert our first pattern. Inversions of melodic themes are very common in musical works by Bach that repeat such themes often, such as his inventions and fugues. See Figure 6.4. We find that the second pattern is more common than the first according to our weight model, as the 20 best matches yield higher weight values, and we also find exact matches of the second pattern in bars 3, 10, 12, and 15—see Figure 6.5. In this case our algorithm happens to properly rank a match of the pattern into the final chord behind the 19 musically sensible matches, but penalties for non-matching note starts would again give a more accurate idea of the quality of that match.

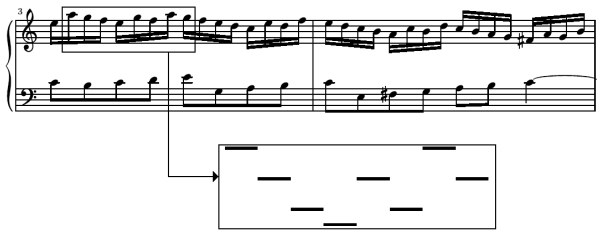


Figure 6.4: Our second pattern, which is an inversion of the first.

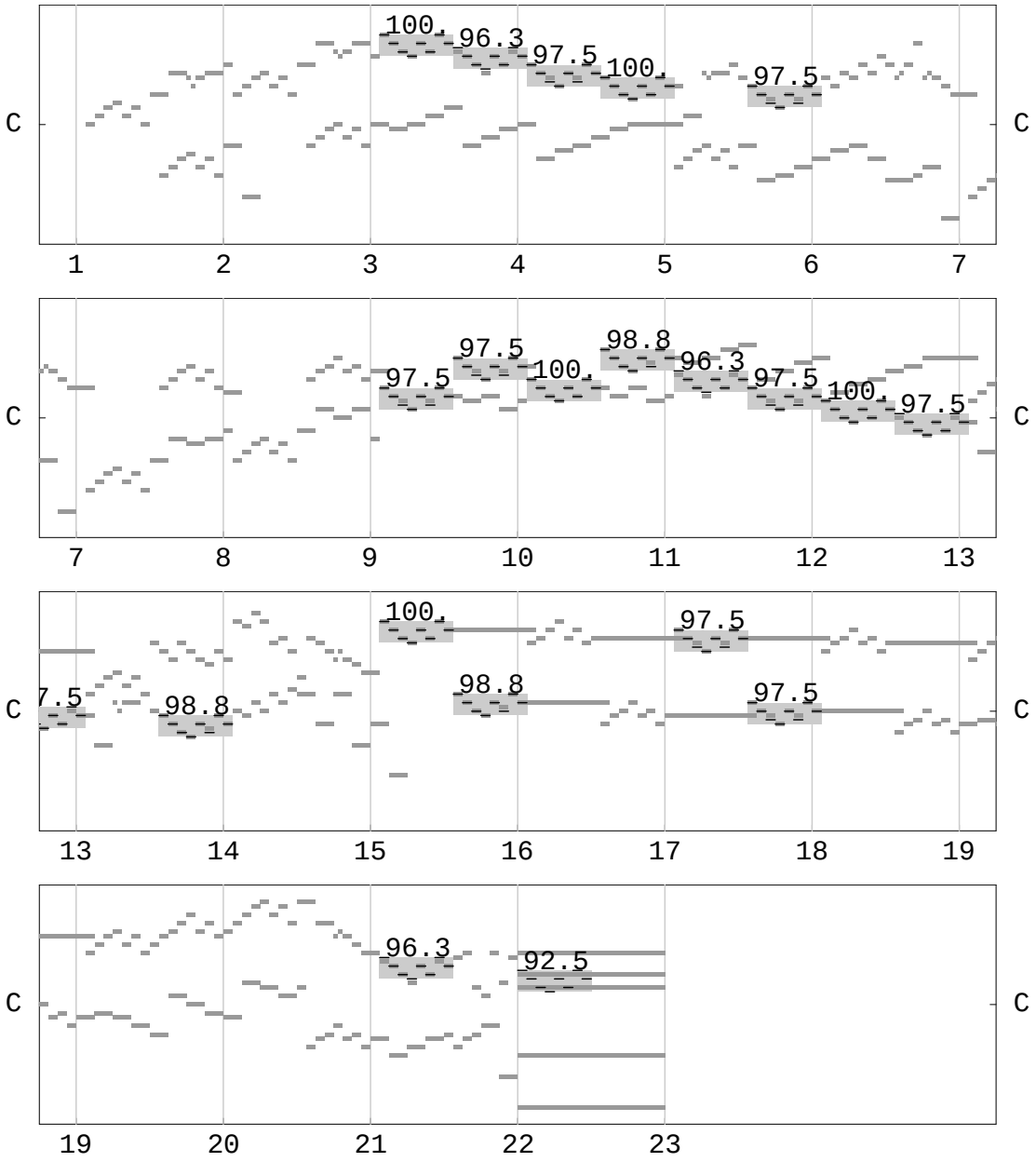


Figure 6.5: The entire line segment representation of the Bach Invention, with matches of our second pattern highlighted.

Our last pattern for this score is taken from another musical work by J. S. Bach. We take the first eight notes played by the right hand in Bach's Prelude in C major from the Well-Tempered Clavier Book II. See Figure 6.6. This pattern consists of seven 16^{th} notes followed by a single 32^{nd} note, and the first three notes of this pattern are exactly an octave higher than the first three notes of the first pattern.

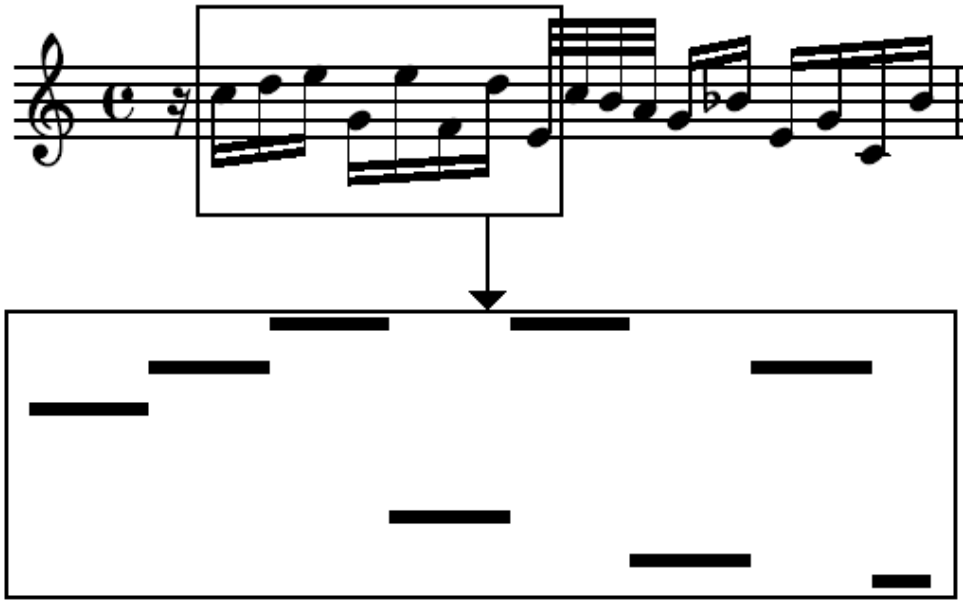


Figure 6.6: Our third pattern, from J. S. Bach's Prelude in C major, Well-Tempered Clavier Book II, bar 1, right hand melody.

Since there is no exact occurrence of this pattern in the score, the quality of matches returned is significantly lower in this case. The five best matches are displayed in Figure 6.7; note that one of the 2^{nd} best matches recorded by our algorithm is the 87.3% false positive that mostly overlaps with the final chord. Although the remaining four matches are of relatively low quality, it is interesting to see how our algorithm matches certain pattern notes to the right-hand part of the invention, and other pattern notes to the left-hand part of the invention. While it is normal for a recognizable monophonic fragment to be played in a single part (whether it is one hand on the piano, or a monophonic instrument), sometimes a monophonic pattern can be recognized as occurring across different parts. Our algorithm, as well as most other algorithms that handle polyphony, can handle such cases well.

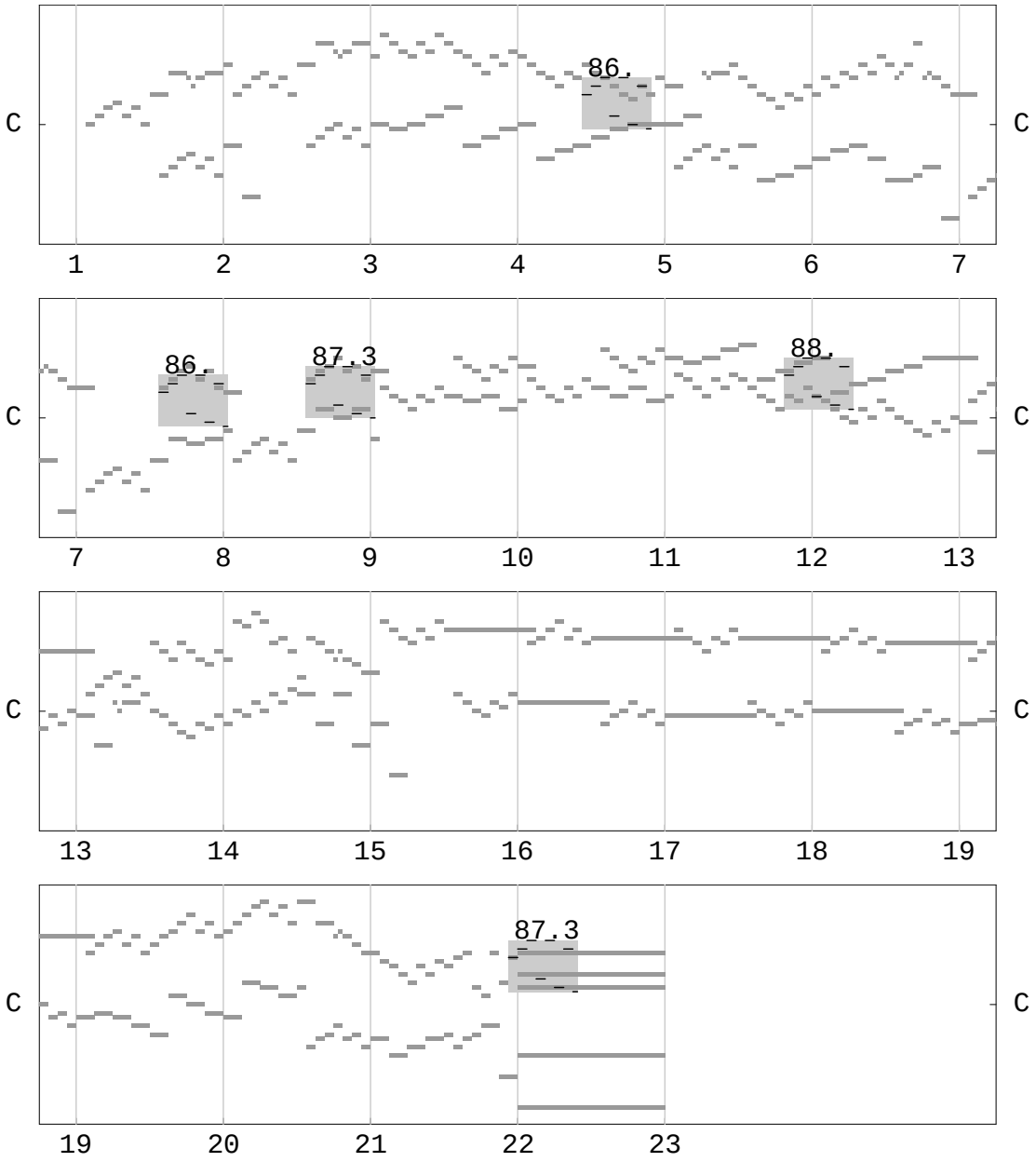


Figure 6.7: The entire line segment representation of the Bach Invention, with matches of our third pattern highlighted.

The second polyphonic score that we examine is the 2$^{\text{nd}}$ movement of Mozart's Piano Sonata K311. The first two bars of this score are shown in Figure 6.8, and in fact our first pattern for this score is taken from the right hand part of these first two bars. This theme

is featured in an example discussed by Selfridge-Field [44]. She observes that the different musical occurrences of the theme in the score vary in different ways, making automatic classification of theme occurrences difficult. Selfridge-Field identifies four iterations of our first pattern: the first in bars 5–6, the second in bars 50–51, the third in bars 54–55, and the fourth in bars 86–87. We will examine how well our algorithm does in identifying the musical occurrences that Selfridge-Field mentions.

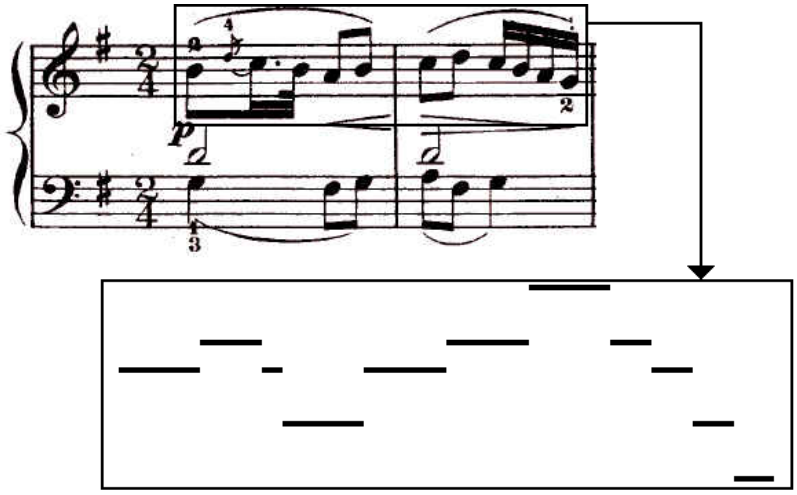


Figure 6.8: The first two bars of W. A. Mozart's Piano Sonata in D major K311, 2nd movement in common musical notation. The line segment representation of our first pattern is shown, omitting the grace note.

This score is significantly longer than the Bach Invention, spanning over 100 bars. In addition, this is the only score in our experiments that was obtained through a MIDI file of a human performance. The main difference between MIDI files derived directly from the score and performance-based MIDI files is that note durations are shorter in the performance-based files. Therefore weight values will be distorted somewhat, and our algorithm may incorrectly assign a good weight to a bad match (or vice versa.)

We discuss the top 20 matches found by our algorithm. The results that highlight the top 11 matches are shown in Figure 6.9. We observe that what should be our best match is ranked second, with a weight of 93.4% instead of 100%. This is due to the performance-based MIDI file, in particular the shortened duration of the score notes. The omission of the grace note from our pattern further reduces the weight. The best nine matches are all good matches, and include all of the occurrences of that theme identified by Selfridge-Field.

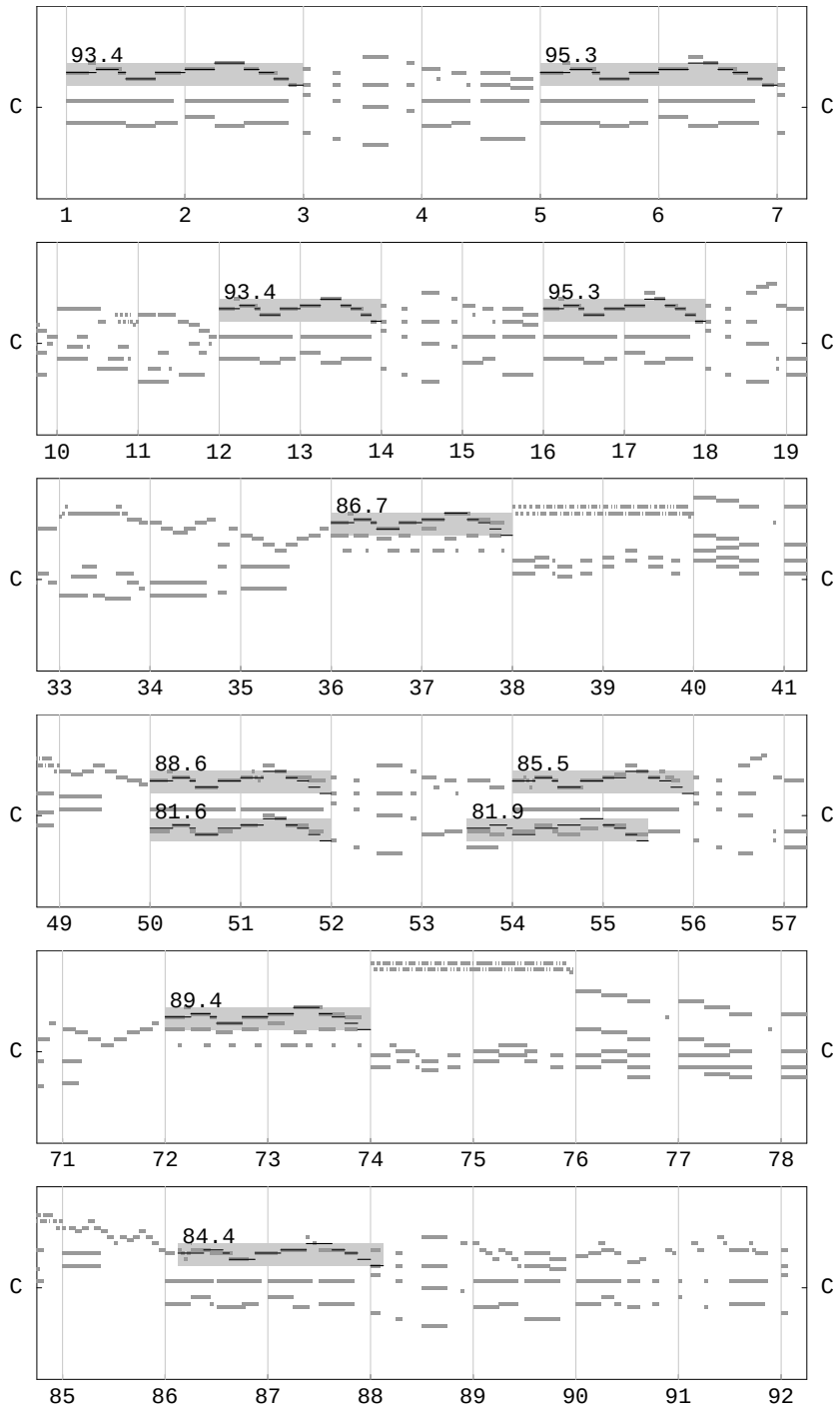


Figure 6.9: The line segment representation of selected bars of the 2nd movement of the Mozart Piano Sonata, with the 11 best matches of our first pattern highlighted.

The 10^{th} best match of the pattern with a weight of 81.9% is a false positive, while the 11^{th} best match with a slightly lower weight of 81.6% does correspond to an occurrence of the theme. This is also most likely due to the differences in the performance-based MIDI file from the score.

Upon further examination of the score, we identify five other occurrences of our pattern in the score that do not even make it into the best 20 matches. Four of these musically sensible matches are not recognized due to the characteristics of the performance-based MIDI file. The theme occurs in the left hand, in thirds, in bars 38–39 and bars 74–75. The performer played these notes staccato, which means that the duration of the notes played is much shorter than would otherwise be the case. The durations of these notes in the MIDI file are roughly half of what the score indicates, and therefore our algorithm produces very low weights around 60%. If we manually change the lengths of these score notes to accurately reflect the music, our algorithm does correctly assign these matches good weights of over 90%.

The last musically sensible match of the first pattern into the score that our algorithm misses is a true challenge. See Figure 6.10. Musically, this is a heavily embellished variant of the theme, but the short note lengths and rests make it difficult for our algorithm to recognize it. This is an example of a match that would be exceedingly difficult for any current music pattern matching algorithm to recognize.

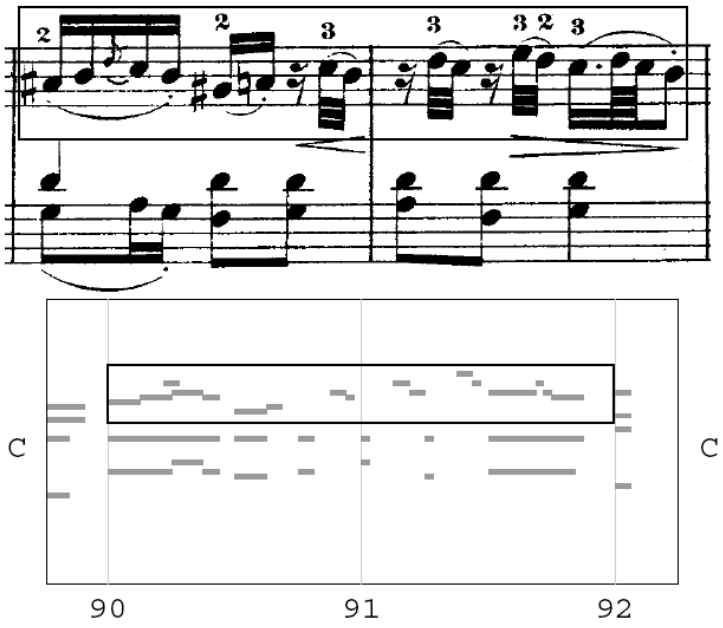


Figure 6.10: Bars 90–91 of W. A. Mozart's Piano Sonata in D major K311, 2^{nd} movement in common musical notation and as line segments. The occurrence of the theme is outlined with a rectangle.

For our second pattern, we use a melodic fragment from the 1st movement of the same piano sonata. See Figure 6.11. Since the 1st movement of the sonata is faster than the 2nd movement, we slow the pattern down in our experiment so that actual note durations are equivalent to those in the 2nd movement. Our algorithm is unable to deal with musical fragments at different speeds as it is currently constructed, and we discuss this limitation further in Section 6.2. Although this pattern does not share any rhythmic similarity to our first pattern, both patterns feature the second, third, fourth, and fifth notes of a major scale prominently.

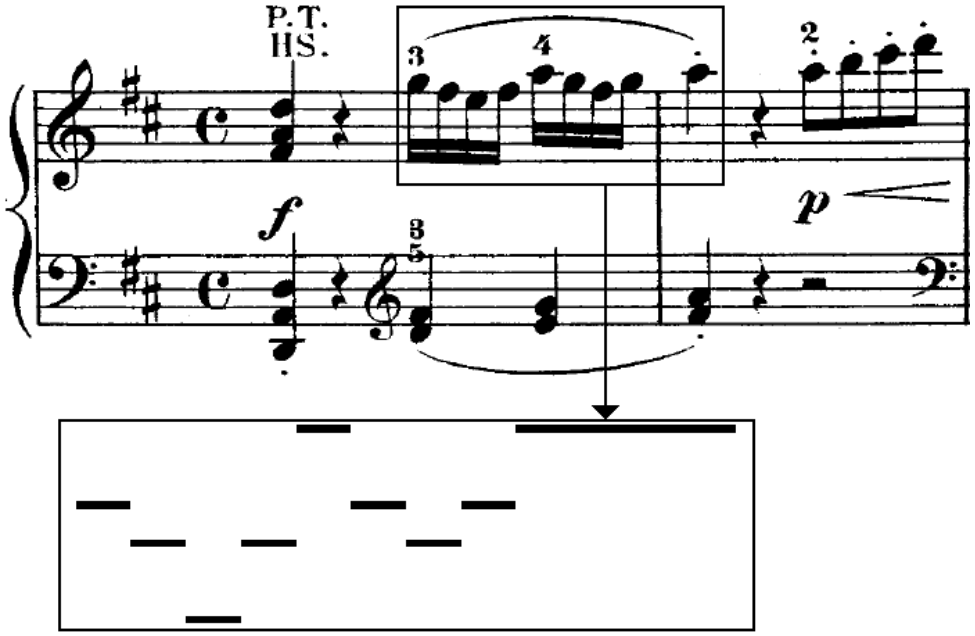


Figure 6.11: The first two bars of W. A. Mozart's Piano Sonata in D major K311, 1st movement in common musical notation. The line segment representation of our second pattern is shown.

We search for the best eight matches of our second pattern into the score, shown in Figure 6.12. Interestingly, the best seven matches correspond roughly to occurrences of the first pattern, where the second pattern is aligned to start at the second note of an occurrence of the first pattern—see Figure 6.13. From bars 49 to 51, we see that the eighth best match, with a weight of 86.7%, appears to "piggyback" on the seventh best match, which has a weight of 86.9%. Since the other six matches start at the same point in the bar as the seventh best match, we conclude that our algorithm correctly recognizes that the eighth best match should be extraneous, according to our particular weight model.

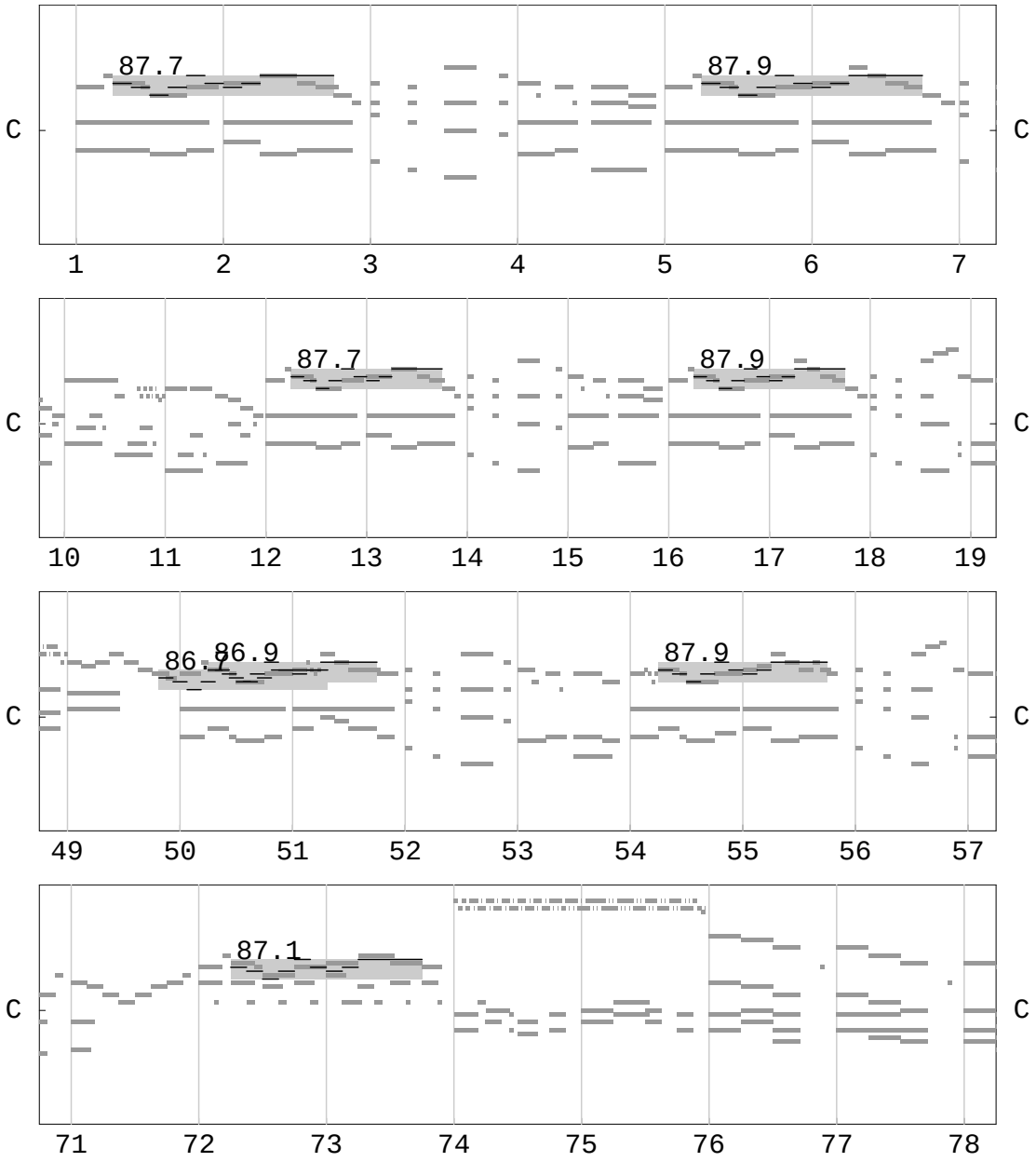


Figure 6.12: The line segment representation of selected bars of the 2nd movement of the Mozart Piano Sonata, with the eight best matches of our second pattern highlighted.

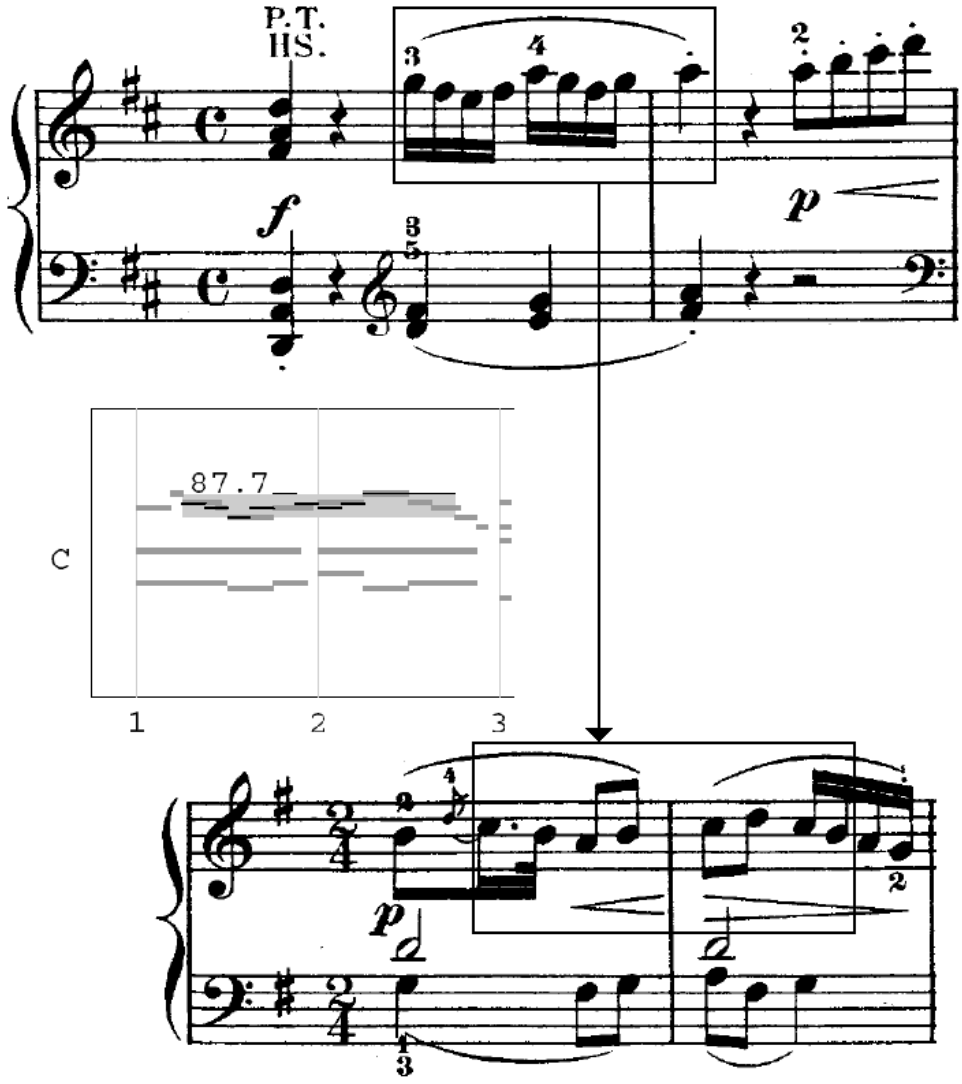


Figure 6.13: Our algorithm matches part of the first two bars of the 1st movement of the Mozart Piano Sonata to part of the first two bars of the 2nd movement.

6.1.2 Polyphonic Patterns

Our first use of polyphonic patterns is applied to the C minor fugue from J. S. Bach's Well-Tempered Clavier, Book I. The length of the fugue is similar to the length of the invention. This fugue consists of three voices, with the main melodic theme being called the *subject*. Each voice performs the subject at different times, with other fairly regular musical conventions governing the interaction between the voices. The first two bars of the piece are shown in Figure 6.14.

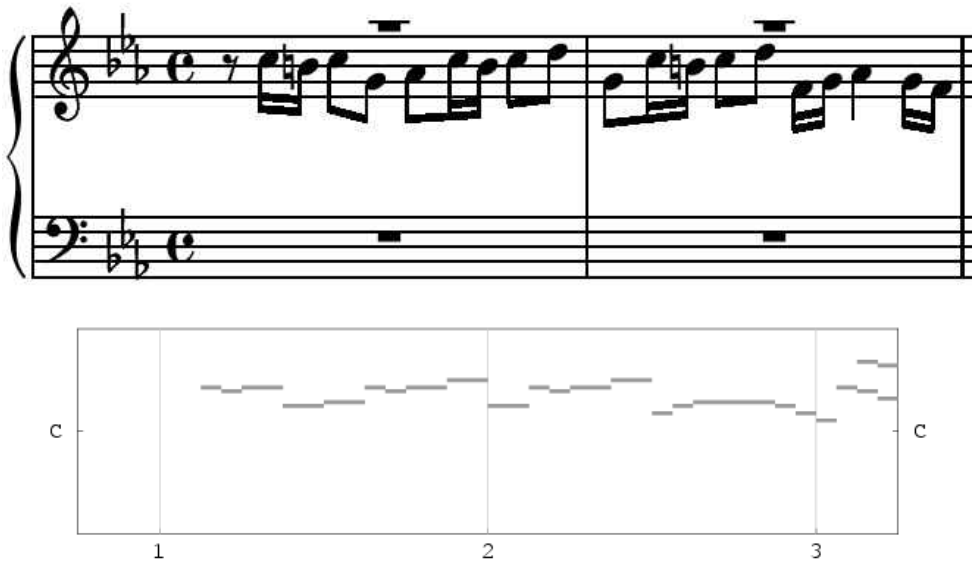


Figure 6.14: The first two bars of J. S. Bach's Fugue in C minor, Well-Tempered Clavier Book I in common musical notation, and in the corresponding line segment representation.

Our polyphonic pattern has an exact occurrence in the score at bar 11—see Figure 6.15. The top voice of this pattern is the first part of the subject transposed into E♭ major, which is the relative major key to C minor. The descending line of 16^{th} notes in the pattern is the counterpoint to the subject, called the *countersubject*, and starts a minor 6^{th} and an additional octave below the E♭ major subject.

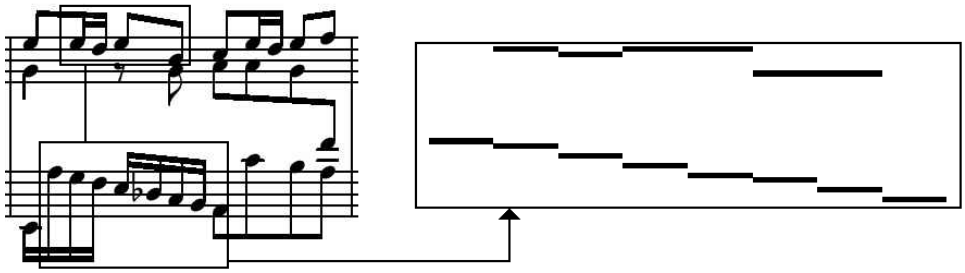


Figure 6.15: Bar 11 of Bach's Fugue in common musical notation. The line segment representation of our polyphonic pattern is shown.

We search for the best 15 matches of our pattern into the score, with the results displayed in Figure 6.16.

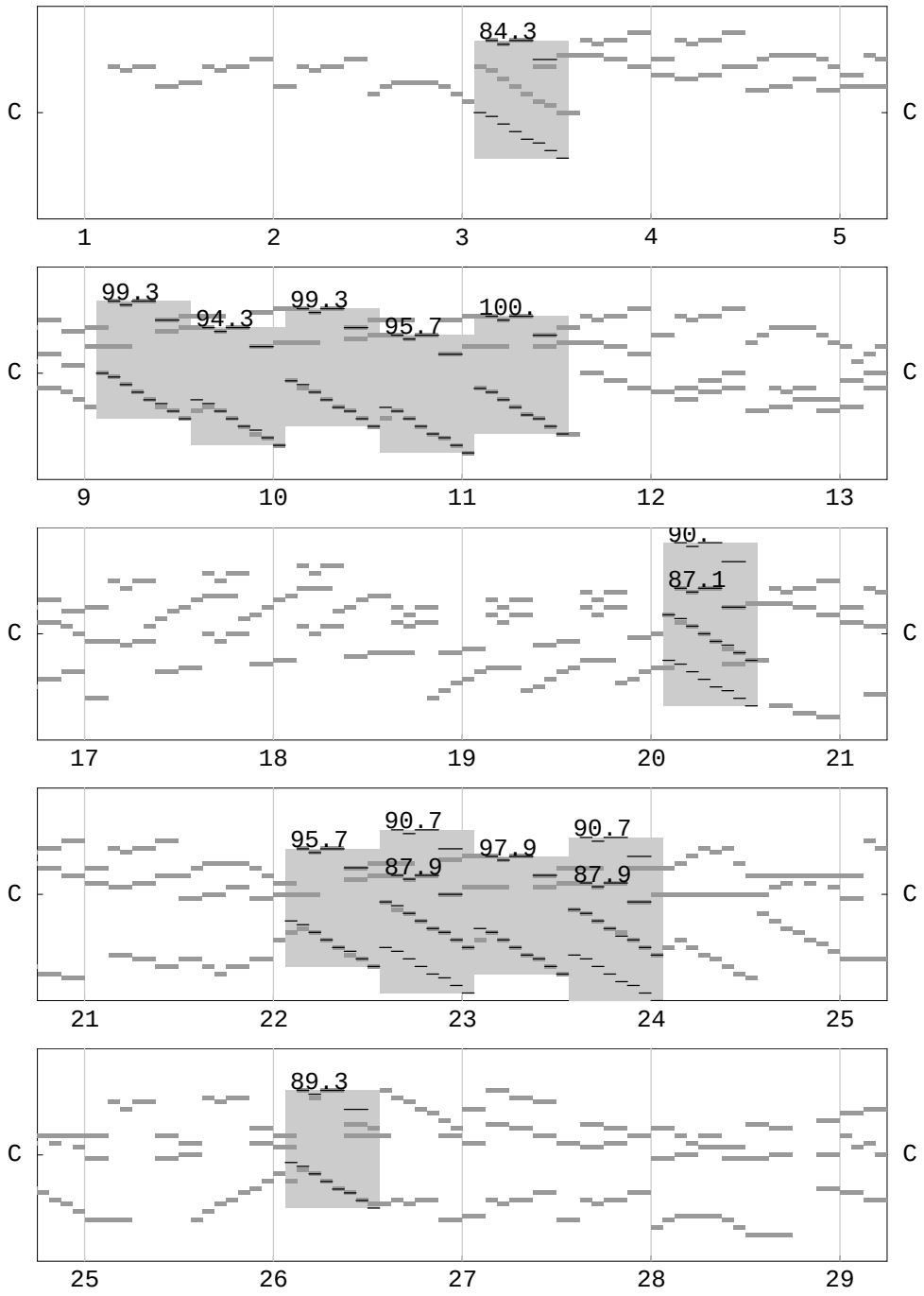


Figure 6.16: The line segment representation of selected bars of J. S. Bach's Fugue in C minor, Well-Tempered Clavier Book I, with the 15 best matches of our polyphonic pattern highlighted.

The best seven matches occur in bars 9–11 and bars 22–23, and have weights above 94%. These are very good matches into the score, and differ from the pattern by a few semitones, at most. Applying some musical analysis, we see that those bars encompass the two developmental episodes in the piece, which modulate to different keys with each repetition of the pattern. Developmental episodes often repeat main fugal themes such as the subject in various ways, and so these matches make a great deal of musical sense.

Our interval-based weight model gives good weights to translated pattern notes that are an octave apart from the corresponding score notes. In our pattern, the voices are a wide interval apart—a minor 6^{th} plus an additional octave. There are occurrences of the pattern in the score where the interval between the two voices is only a minor 6^{th}. Our algorithm successfully finds such matches; for example, see the 84.3% match in bar 3 of the previous figure. Matches of this type also occur in the second halves of bar 22 and 23. These latter matches occur twice; once with the top voice being an octave higher than the corresponding score notes, and once with the bottom voice being an octave lower than the corresponding score notes. This is logical, and we call the match with the lower weight at the same horizontal translation a "phantom" match. (The lower weight occurs as a result of the notes in the bottom voice having a greater cumulative duration than the notes in the top voice, and other interval-based differences that are not always exactly an octave apart.) Note that the phantom partner for the match that occurs in bar 3 has a lower weight that falls below our threshold of the 15 best matches. Our algorithm properly distinguishes good matches from poor matches, apart from these phantom occurrences.

Our last use of polyphonic patterns is applied to the 2^{nd} movement of Haydn's "Emperor" String Quartet in C major Opus 76, Number 3. The length of this movement is quite long, and is comparable to the length of the 2^{nd} movement of the Mozart Piano Sonata. The string quartet consists of four parts, corresponding to two violins, one viola, and one cello. The format of this movement consists of a theme and four variations, and the repetition that occurs as a result of this format is well-suited for our algorithm. The first two bars of the piece are shown in Figure 6.17.

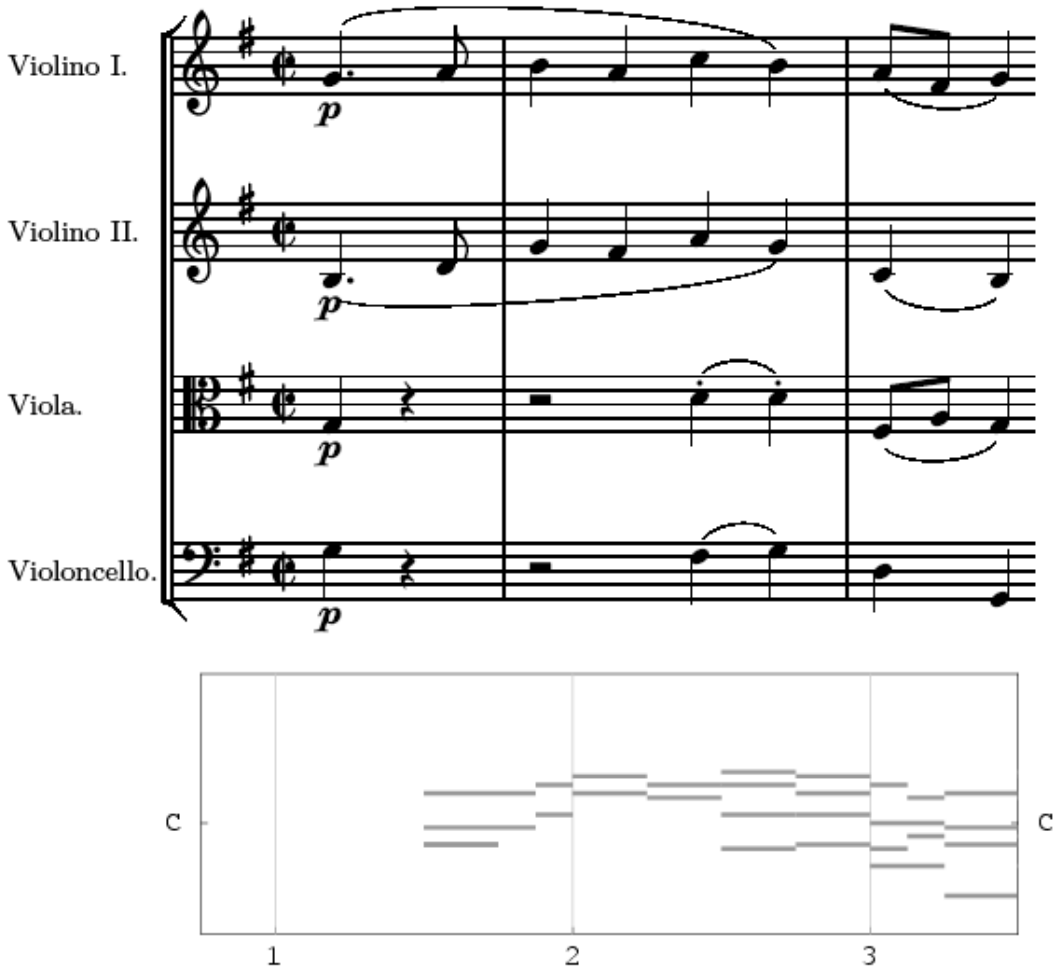


Figure 6.17: The first two bars of J. Haydn's "Emperor" String Quartet in C major Op. 76 No. 3, 2nd movement in common musical notation, and in the corresponding line segment representation.

Our polyphonic pattern is taken from the first few bars of Variation II, with the top voice performed by the first violin, the middle voice performed by the second violin, and the bottom voice performed by the cello. The cello plays the predominant melodic theme in the piece. See Figure 6.18. Since other occurrences of this theme take place at higher pitches, it is unlikely that the pattern will match well to these occurrences because the top two voices of the pattern are unlikely to have anything to match to. Therefore we are interested in seeing other places in the score where this complex polyphonic pattern will match well.

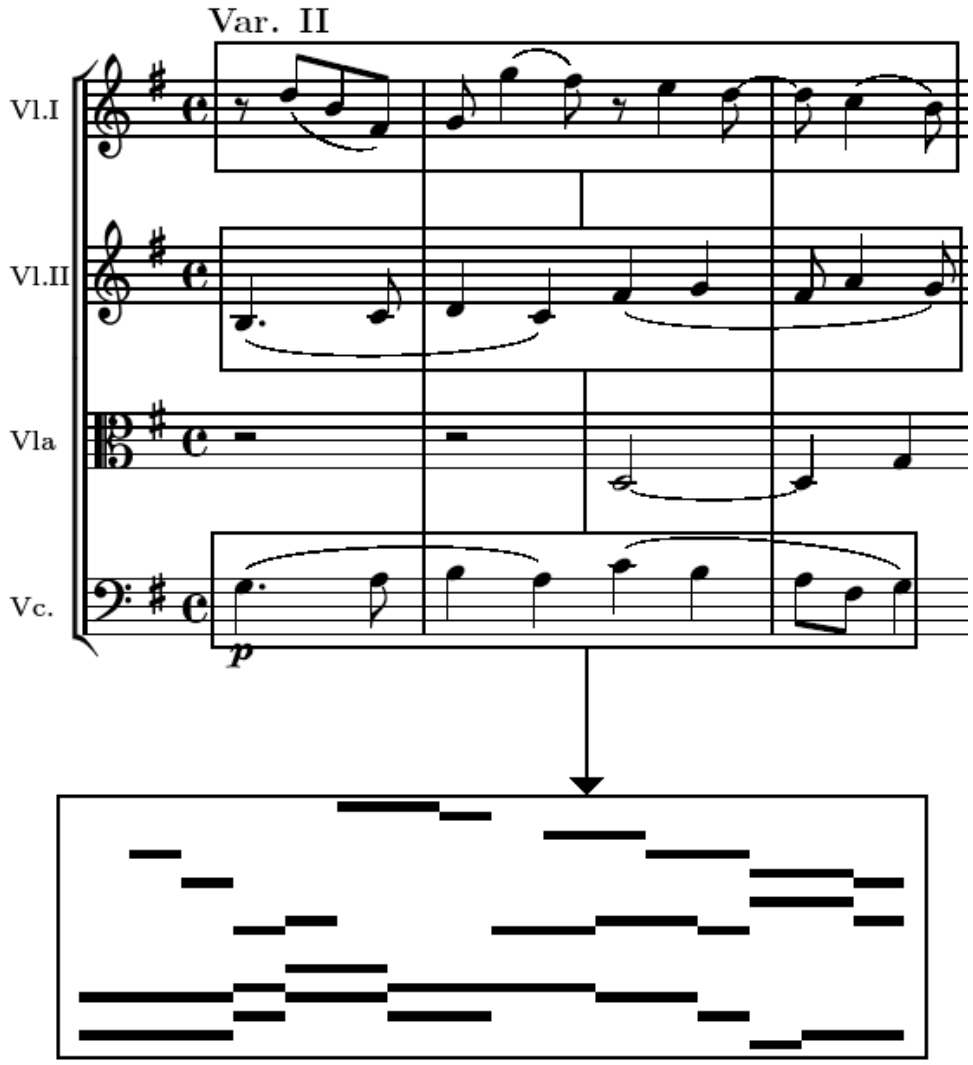


Figure 6.18: Our polyphonic pattern, which takes place in the first violin, second violin, and cello parts at the start of Variation 2.

We search for the best eight matches of our pattern into the score, with the results displayed in Figure 6.19.

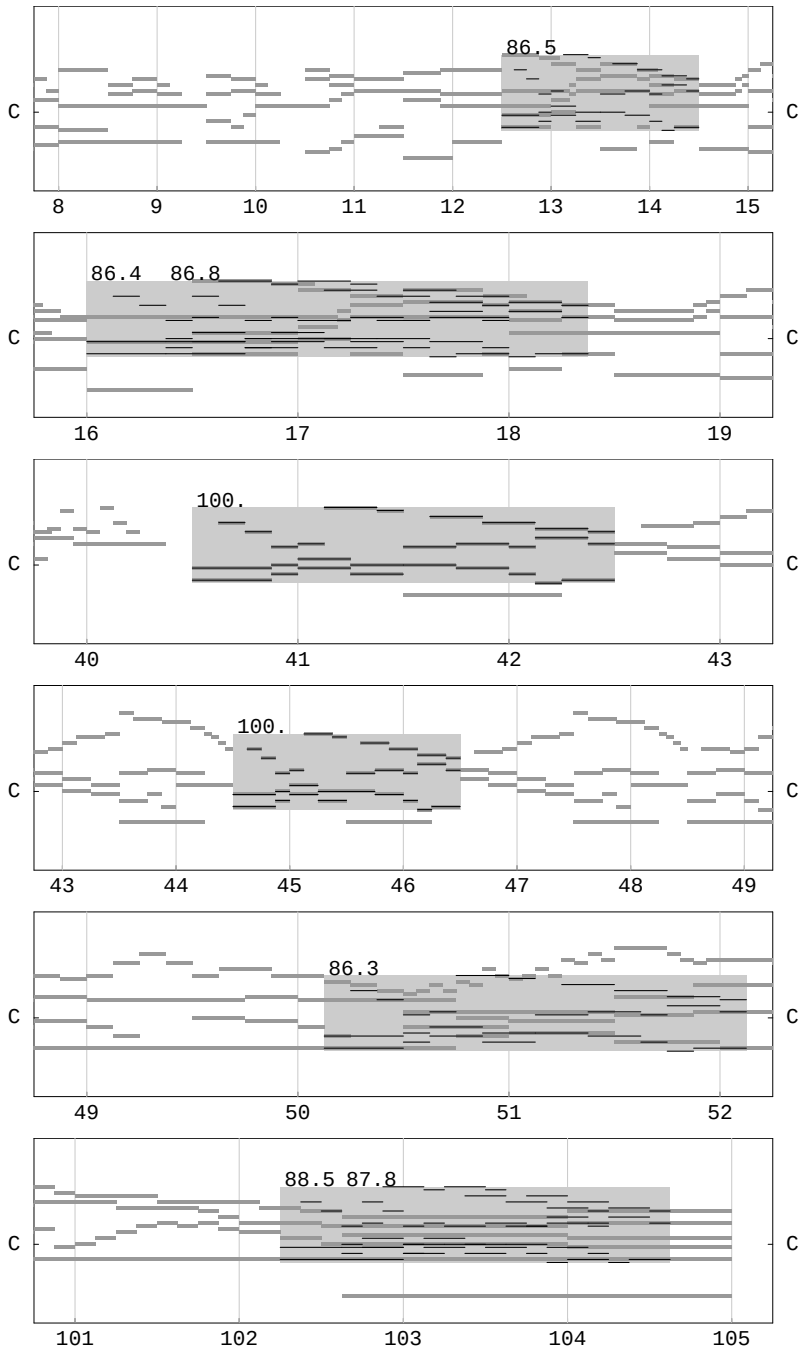


Figure 6.19: The line segment representation of selected bars of the 2nd movement of Haydn's String Quartet, with the eight best matches of our polyphonic pattern highlighted.

The theme and each variation except the last are 20 bars long, with the fourth and final variation lasting 24 bars. Thus Variation I starts in the middle of bar 20, Variation II starts in the middle of bar 40, Variation III starts in the middle of bar 60, and Variation IV starts in the middle of bar 80. We find three matches of the pattern towards the end of the theme, one in bars 12–14, and two overlapping matches in bars 16–18. These matches are interesting, since the pattern originates from the start of Variation II—thus we would expect that the pattern might match better to the start of the theme, or any of the other variations, rather than the end of the theme. While it would be wrong to classify these matches as occurrences of the pattern, from a music theoretic point of view, there is at least some musical similarity between the multiple voices at those points—see Figure 6.20.

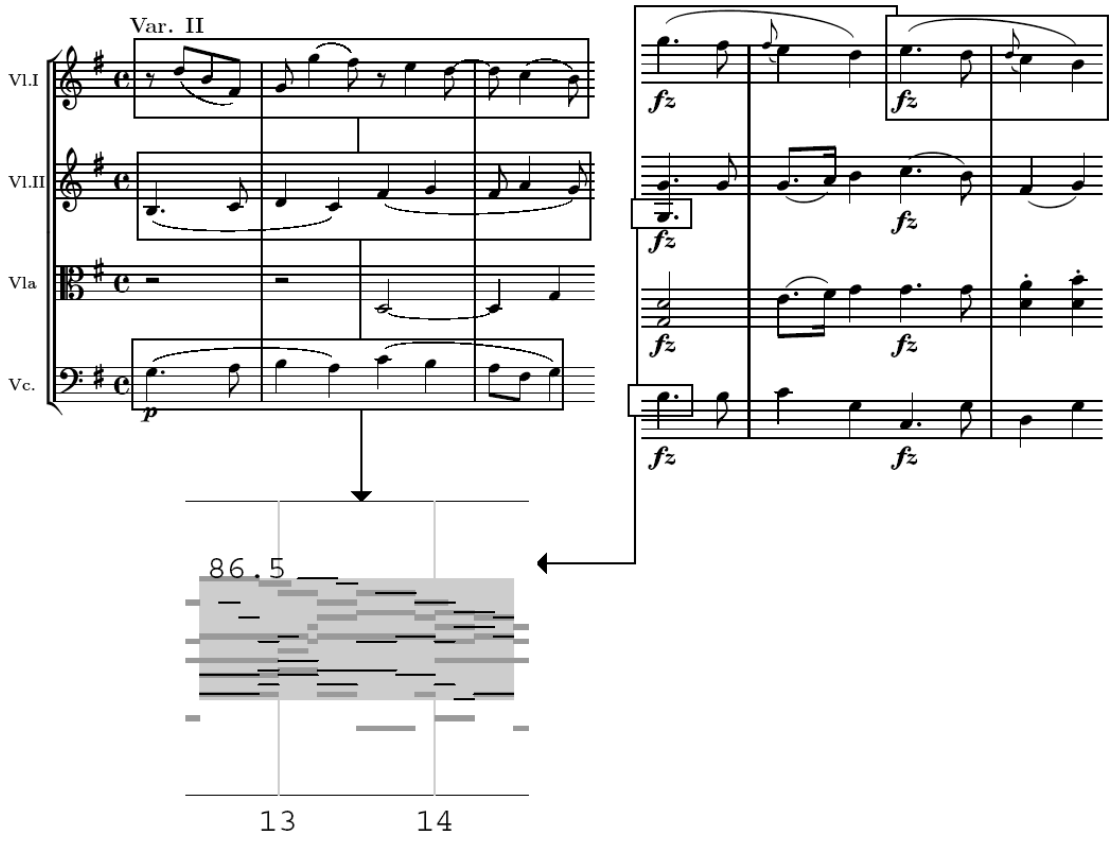


Figure 6.20: A match of our polyphonic pattern into bars 12–14 of the theme. Note the similarities in the second half of the top voice, as well as the common notes in the bottom and middle voices at the start of the match.

Note also that these matches happen to match pattern notes from the 2nd violin part to the cello, similar to how our last pattern for the Bach Invention matched some pattern notes to the left hand part, and others to the right hand part on the piano. In this case, a violin has a different timbre than a cello, and therefore it may be musically sensible to apply some sort of penalty for mismatching timbres. This highlights a limitation of a representation that only takes pitch and duration into account.

Two exact matches of the pattern occur in Variation II, at bars 40–42 and bars 44–46. There is one other approximate match in bars 50–52. This also cannot be classified as an occurrence of the pattern, but this match does highlight some degree of interval-based similarity. The remaining two matches of the pattern overlap the final chords at the end of the piece, again demonstrating why giving more weight to matching the starts of notes would be a good idea. This particular experiment demonstrates the difficulty of finding musically sensible matches when the pattern has a high degree of polyphony, which makes sense. This pattern also uses a wide range of pitches, spanning two octaves. This limits the potential number of matches compared to a pattern with a shorter range of pitches, because transpositions that match pattern notes at one pitch extreme to the score may not match pattern notes at the other pitch extreme at all.

6.2 Discussion

These experiments demonstrate that our algorithm can be used to find musically sensible matches of a pattern into a polyphonic score, given an appropriate interval-based weight model. For the Bach Invention, we examined how our algorithm correctly finds a large number of approximate matches of two patterns that have exact matches into the score, and also showed how our algorithm could match different parts of a monophonic pattern to either of the two parts in the score. Looking at the Mozart Piano Sonata, we dealt with additional difficulties resulting from the use of a MIDI file based on a human performance, but still managed to recognize many elaborate melodic variations of our pattern.

Examining polyphonic patterns, we see that musically sensible matches of the polyphonic pattern into the Bach C minor Fugue can still achieve a good weight even if some of the pattern notes are an octave apart from the corresponding score notes. These are excellent examples of matches that yield good weights according to our interval-based weight model, while a distance-based weight model would not recognize the good quality of these matches. Lastly, we see some interesting results when examining the matches in the Haydn String Quartet, and highlights some of the natural difficulties of finding good matches when using a pattern with a high degree of polyphony.

One significant shortcoming of our algorithm is that it is completely dependent on the duration of the pattern notes. For example, a musically sensible occurrence of the pattern in the score may be played twice as fast, or twice as slow compared to our line segment representation of our pattern. Our algorithm will not recognize such occurrences,

because their durations will be different from the duration of the pattern. See Figure 6.21. However, our algorithm can handle minor rhythmic variations of the pattern, as shown in the Mozart example. The key there is that the overall duration of the pattern does not change.

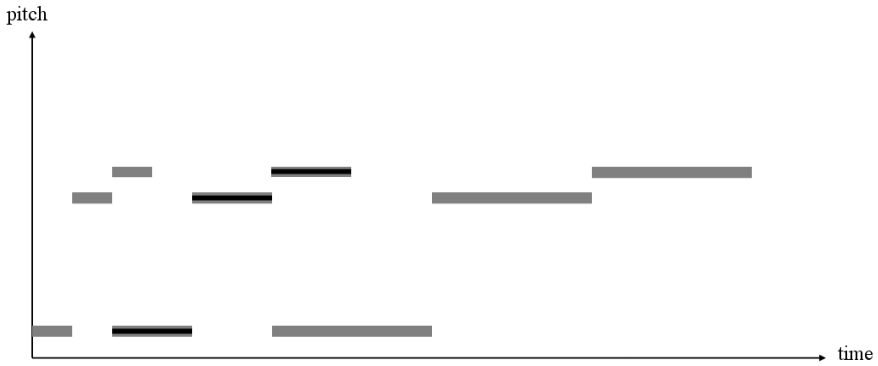


Figure 6.21: A score consisting of three occurrences of a two note pattern at different speeds. The pattern only matches exactly to the middle occurrence.

As seen in the previous section, another side-effect of our algorithm's dependence on the duration of the pattern is that occurrences of the pattern in the score with shortened notes are not given good weights. While this issue usually arises with performance-based MIDI files, it can also occur with MIDI files that are automatically generated from the score. In Figure 6.22 we see that there is an occurrence of the pattern in the score that receives a weight of 50%, simply because each note in the score has half the duration of the corresponding pattern note, and there are rests between each score note.

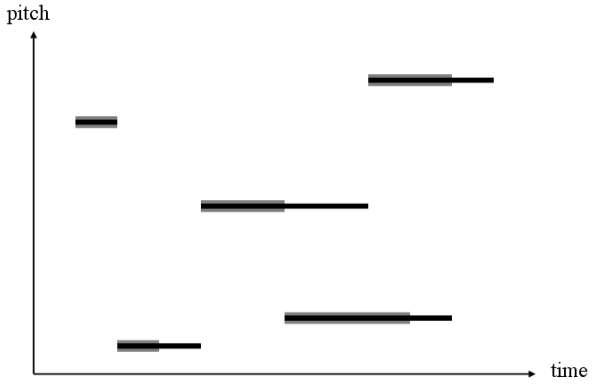


Figure 6.22: An occurrence of a pattern in a score that receives a poor weight due to the note durations.

Note that the reverse case results in a weight of 100%; that is, if the pattern consists of the shorter notes with rests in between, and the score consists of the long notes, simply lining the note starts up results in an exact match of the pattern into the score. This is undesirable, in that we would prefer a weight function that is symmetric with respect to the translated pattern and the relevant part of the score. For example, the reverse case would receive a weight lower than 100% if we assigned some sort of penalty for rests in the pattern matching to notes in the score. We can also find a weight greater than 50% for the first case if we assigned greater weight for matching the starts of pattern notes to starts of score notes.

Our algorithm also has other asymmetries. We have already briefly discussed assigning different weights to similar intervals such as the perfect 4^{th} and the perfect 5^{th}, although the same note pitches may be involved in both cases, with an octave difference for one of the notes. This may cause our algorithm to miss potentially interesting matches where matching score notes range both above and below pattern notes, or vice versa. However, this does not affect most of the matches that our algorithm encounters, where the score notes matched to a pattern are mostly of higher (or lower) pitch than the corresponding pattern notes.

Allowing more than one pattern note to match to the same score note might not make sense when examining polyphonic music that is composed of multiple monophonic voices, unless multiple score notes take place at the same time. For more general polyphonic music, where the polyphony changes over time, this asymmetry may make more sense.

As seen in the Bach Invention and the Haydn String Quartet, allowing more weight for note starts also deals with the false positives that match a pattern with a limited pitch range into long chords in the score. Therefore it seems that implementing the extension to our algorithm to keep track of note starts will help us on two fronts: the false positives of this type will be eliminated because pattern notes matching into the middle of score notes will receive less weight, and the penalty for matching a pattern note to a score note of shorter duration would be reduced. As discussed in Subsection 4.7.3, implementing greater weights for matching note starts is possible, and does not increase the running time of our algorithm.

This still leaves us with the problem of finding matches of the pattern in the score that are either slower or faster than the original pattern. The approach to handle this in other music pattern matching algorithms is to include duration ratios in the representation of the pattern and score. Similar to the concept of transposition, which ensures that we are only concerned with relative pitches when determining similarity instead of absolute pitches, duration ratios deal with relative durations. Therefore an occurrence of a pattern played at a different speed will have an identical duration ratio to that of the original pattern.

We do not see how to use duration ratios without increasing the running time of our algorithm. Although it would be nice to be able to detect occurrences of a pattern in

a score that have different speeds compared to the original pattern, such occurrences are usually limited to certain types of Baroque music, as well as Renaissance music. Occurrences of the pattern that differ in terms of pitch and local note durations are far more common than occurrences of the pattern that differ in terms of total duration, in many cases.

Chapter 7

Conclusions and Future Work

We have presented an approach to the music pattern matching problem that uses a natural geometric representation. This allows us to solve the problem of finding a musically sensible match of a pattern into a score by finding the best translation of a small set of horizontal line segments into a larger set of horizontal line segments. While approaches using this same geometric framework have been explored in the past, we are able to contribute a powerful way of finding musically sensible matches in polyphonic music by using general weight models.

Through our experiments, we have only started to demonstrate what might be possible by using a general interval-based weight model. Since different arguments can be made for different interval-based weights, based on certain types of music, for example, this thesis provides a solid framework for exploring how various interval-based weights can be applied to polyphonic music pattern matching.

As shown in Section 4.2, our algorithm has comparable efficiency to previous algorithms. While our running time's dependence on the size of our discrete pitch set is undesirable, the ability of our algorithm to achieve complex, musically sensible ways of music pattern matching compensates for this.

There are many avenues for future work. We discussed giving additional weights to matching note starts in Subsection 4.7.3, and found evidence that doing so would lead to more musically sensible matches. More rigourous experiments are also needed to refine musically sensible weight models, and also to apply a proper implementation to a larger collection of data. Ideally we can use learning methods on a set of training weights, in order to obtain a good set of interval-based weights.

Incorporating different types of musical information into our algorithm would also be a useful direction for future research. Handling information about note stresses (which can be strong or weak depending on the position of the note in the bar), or about the key

of tonal music, may possibly be achieved without drastically changing our pitch/duration-based representation. Adding information about other musical features such as dynamics, timbre, or articulation would probably require an additional dimension to be added to our line segment representation, while increasing the complexity of calculating the weight of a match. Of these other musical features, timbre seems like a good candidate for yielding more musically sensible matches when many different instruments have parts in the score.

It will also be beneficial to explore the possibility of richer harmonic comparisons between the pattern and the score. In our current framework, we compare one note of the pattern to one note of the score at a particular time and examine the interval formed. This is restrictive, in the sense that other pattern and score notes being played at the same time can also interact with each other to form a particular chord. Therefore comparing the chord formed by pattern notes at a certain time to the chord formed by score notes at that time may lead to more musically meaningful matches.

As discussed in Section 6.2, incorporating duration ratios into our algorithm to handle matches of the pattern in the score that are either faster or slower than the original pattern will be a challenge. While duration ratios can be useful, we do not see how the time point framework used in our algorithm can be accommodated.

Our algorithm does not explicitly deal with rests. Currently there is no weight penalty if rests in the pattern match to notes in the score, and therefore there is no benefit if rests in the pattern match to rests in the score. It may make sense for weight penalties to be applied to differentiate between these two situations, and it would be interesting to attempt to incorporate this into our algorithm. Matching pattern notes to score rests gives zero weight for those pattern notes, while matching pattern rests to score notes does not affect the weight. Related to this is the issue of matching multiple notes in the pattern to a single note in the score.

There are at least two possible ways to improve the efficiency of our algorithm. Noting that m is usually quite small compared to d (especially for the set of MIDI pitches), lessening our running time's dependency on d would be desirable. In practice, there are often many vertical transpositions that do not need to be checked. If we can somehow determine whether a particular transposition is irrelevant, we will be able to save a great deal of time.

Another way to improve efficiency is to trade off some degree of correctness. In many other algorithmic fields, heuristics exist that generally find the best solution to a problem with some high probability, but take significantly less time to do so when compared to an algorithm that solves the problem exactly. It will be interesting to see if this heuristic approach can be applied to the music pattern matching domain.

Bibliography

[1] G. Aloupis, T. Fevens, S. Langerman, T. Matsui, A. Mesa, D. Rappaport, and G. Toussaint. Computing the similarity of two melodies. In *Proceedings of the 15th Canadian Conference on Computational Geometry*, pages 81–84, 2003.

[2] D. Bainbridge, D. Byrd, T. Crawford, J. S. Downie, J. Dunn, M. Fingerhut, I. Fujinaga, H. Hoos, K. Lemström, and C. Mayer. The international conferences on music information retrieval (ISMIR) and related activities. http://www.ismir.net/.

[3] G. Barequet and S. Har-Peled. Polygon-containment and translational min-Hausdorff-distance between segment sets are 3SUM-hard. *International Journal of Computational Geometry and Applications*, 11(4):465–474, 2001.

[4] R. S. Boyer and J. S. Moore. A fast string searching algorithm. *Communications of the ACM*, 20:762–772, October 1977.

[5] A. Brinkman and M. Mesiti. Graphic modeling of musical structure. *Computers in Music Research*, 3:1–42, 1991.

[6] D. Byrd and T. Crawford. Problems of music information retrieval in the real world. *Information Processing and Management*, 38:249–272, 2002.

[7] L. Chiariglione. Technology and art – putting things in context. In *Proceedings of the 3rd International Conference on Music Information Retrieval (ISMIR 2002)*, pages 238–240, 2002.

[8] M. Cooper and J. Foote. Automatic music summarization via similarity analysis. In *Proceedings of the 3rd International Conference on Music Information Retrieval (ISMIR 2002)*, pages 81–85, 2002.

[9] P. J. O. Doets and R. L. Lagendijk. Stochastic model of a robust audio fingerprinting system. In *Proceedings of the 5th International Conference on Music Information Retrieval (ISMIR 2004)*, pages 349–352, 2004.

[10] J. S. Downie. Music information retrieval. *Annual Review of Information Science and Technology*, 37:295–340, 2003.

[11] J. S. Downie. Toward the scientific evaluation of music information retrieval systems. In *Proceedings of the 4th International Conference on Music Information Retrieval (ISMIR 2003)*, pages 25–32, 2003.

[12] J. S. Downie, K. Medina, A. Enright, J. Seymour, and R. Ramsey. Music information retrieval annotated bibliography. http://www.music-ir.org/research_home.html.

[13] J. S. Downie and M. Nelson. Evaluation of a simple and effective music information retrieval method. In *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 73–80, 2000.

[14] A. Efrat, P. Indyk, and S. Venkatasubramanian. Pattern matching for sets of segments. In *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2001)*, pages 295–304, 2001.

[15] J. Eggink and G. J. Brown. Application of missing feature theory to the recognition of musical instruments in polyphonic audio. In *Proceedings of the 4th International Conference on Music Information Retrieval (ISMIR 2003)*, pages 125–131, 2003.

[16] J. Foote. An overview of audio information retrieval. *Multimedia Systems*, 7:2–11, 1999.

[17] C. Francu and C. G. Nevill-Manning. Distance metrics and indexing strategies for a digital library of popular music. In *Proc. IEEE International Conference on Multimedia and EXPO (II)*, pages 889–894, 2000.

[18] J. Futrelle and J. S. Downie. Interdisciplinary research issues in music information retrieval: ISMIR 2000–2002. *Journal of New Music Research*, 32:121–131, June 2003.

[19] A. Gajentaan and M. H. Overmars. On a class of $o(n^2)$ problems in computational geometry. *Computational Geometry:Theory and Applications*, 5(3):165–185, 1995.

[20] M. Goto, K. Itou, K. Kitayama, and T. Kobayashi. Speech-recognition interfaces for music information retrieval: 'speech completion' and 'speech spotter'. In *Proceedings of the 5th International Conference on Music Information Retrieval (ISMIR 2004)*, pages 403–408, 2004.

[21] D. Gusfield. *Algorithms on strings, trees, and sequences: computer science and computational biology.* Cambridge University Press, New York, NY, USA, 1997.

[22] W. B. Hewlett. A base-40 number line representation of musical pitch notation. *Musikometrika*, 50:1–14, 1992.

[23] J. Hsu, A. L. P. Chen, H. Chen, and N. Liu. The effectiveness study of various music information retrieval approaches. In *Proceedings of the 11th International Conference on Information and Knowledge Management (CIKM 2002)*, pages 422–429, 2002.

[24] Ipsos-Reid. Popularity of fee-based music downloading takes off. http://www.ipsos-na.com/news/pressrelease.cfm?id=2550.

[25] M. Kassler. Toward musical information retrieval. *Perspectives of New Music*, 4:59–67, 1966.

[26] D. E. Knuth, J. H. Morris, and V. R. Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, 6:323–350, June 1977.

[27] O. Lartillot. Discovering musical patterns through perceptive heuristics. In *Proceedings of the 4th International Conference on Music Information Retrieval (ISMIR 2003)*, pages 89–96, 2003.

[28] J. H. Lee and J. S. Downie. Survey of music information needs, uses, and seeking behaviours: preliminary findings. In *Proceedings of the 5th International Conference on Music Information Retrieval (ISMIR 2004)*, pages 441–446, 2004.

[29] K. Lemström. *String Matching Techniques for Music Retrieval*. PhD thesis, University of Helsinki, Department of Computer Science, 2000.

[30] K. Lemström and J. Tarhio. Transposition invariant pattern matching for multi-track strings. *Nordic Journal of Computing*, 10:185–205, 2003.

[31] K. Lemström, G. A. Wiggins, and D. Meredith. A three-layer approach for music retrieval in large databases. In *Proceedings of the 2nd International Symposium on Music Information Retrieval (ISMIR 2001)*, pages 13–14, 2001.

[32] B. Logan. Music recommendation from song sets. In *Proceedings of the 5th International Conference on Music Information Retrieval (ISMIR 2004)*, pages 425–428, 2004.

[33] A. Lubiw and L. Tanur. Pattern matching in polyphonic music as a weighted geometric translation problem. In *Proceedings of the 5th International Conference on Music Information Retrieval (ISMIR 2004)*, pages 289–296, 2004.

[34] D. Ó Maidín. A geometrical algorithm for melodic difference. In W. B. Hewlett and E. Selfridge-Field, editors, *Melodic Similarity: Concepts, Procedures, and Applications*, volume 11 of *Computing in Musicology*, pages 65–72. MIT Press, 1997–1998.

[35] S. Malinowski. Music animation machine. http://www.well.com/user/smalin/mam.html.

[36] M. F. McKinney and J. Breebaart. Features for audio and music classification. In *Proceedings of the 4th International Conference on Music Information Retrieval (ISMIR 2003)*, pages 151–158, 2003.

[37] M. Mongeau and D. Sankoff. Comparison of musical sequences. *Computers and the Humanities*, 24:161–175, 1990.

[38] H. W. Nienhuys, J. C. Nieuwenhuizen, et al. GNU LilyPond: an automated engraving system. http://www.lilypond.org/.

[39] Recording Industry Association of America (RIAA). RIAA's 2004 yearend statistics. http://www.riaa.com/news/newsletter/pdf/2004yearEndStats.pdf.

[40] T. A. Olson and J. S. Downie. Chopin early editions: construction and usage of online digital scores. In *Proceedings of the 4th International Conference on Music Information Retrieval (ISMIR 2003)*, pages 247–248, 2003.

[41] K. S. Orpen and D. Huron. Measurement of similarity in music: a quantitative approach for non-parametric representations. *Computers in Music Research*, 4:1–44, 1992.

[42] S. Pauws. Musical key extraction from audio. In *Proceedings of the 5th International Conference on Music Information Retrieval (ISMIR 2004)*, pages 96–99, 2004.

[43] C. Sawer and D. Chan. The mutopia project. http://www.mutopiaproject.org.

[44] E. Selfridge-Field. Conceptual and representational issues in melodic comparision. In W. B. Hewlett and E. Selfridge-Field, editors, *Melodic Similarity: Concepts, Procedures, and Applications*, volume 11 of *Computing in Musicology*, pages 3–64. MIT Press, 1997–1998.

[45] A. Sheh and D. P. W. Ellis. Chord segmentation and recognition using EM-trained hidden Markov models. In *Proceedings of the 4th International Conference on Music Information Retrieval (ISMIR 2003)*, pages 183–189, 2003.

[46] J. Shifrin and W. Birmingham. Effectiveness of HMM-based retrieval on large databases. In *Proceedings of the 4th International Conference on Music Information Retrieval (ISMIR 2003)*, pages 33–39, 2003.

[47] J. Shifrin, B. Pardo, C. Meek, and W. P. Birmingham. HMM-based musical query retrieval. In *Proceedings of the ACM International Joint Conference on Digital Libraries (JCDL)*, pages 295–300, 2002.

[48] R. P. Smiraglia. Musical works as information retrieval entities: epistemological perspectives. In *Proceedings of the 2nd International Symposium on Music Information Retrieval (ISMIR 2001)*, pages 85–91, 2001.

[49] R. Typke, P. Giannopoulos, R. C. Veltkamp, F. Wiering, and R. van Oostrum. Using transportation distances for measuring melodic similarity. In *Proceedings of the 4th International Conference on Music Information Retrieval (ISMIR 2003)*, pages 107–114, 2003.

[50] A. L. Uitdenbogerd and Y. W. Yap. Was Parsons right? An experiment in usability of music representations for melody-based music retrieval. In *Proceedings of the 4th International Conference on Music Information Retrieval (ISMIR 2003)*, pages 75–79, 2003.

[51] E. Ukkonen, K. Lemström, and V. Mäkinen. Geometric algorithms for transposition invariant content-based music retrieval. In *Proceedings of the 4th International Conference on Music Information Retrieval (ISMIR 2003)*, pages 193–199, 2003.

[52] R. Wagner and M. Fischer. The string-to-string correction problem. *Journal of the ACM*, 21:168–173, January 1974.

[53] G. A. Wiggins, K. Lemström, and D. Meredith. SIA(M)ESE: An algorithm for transposition invariant, polyphonic content-based music retrieval. In *Proceedings of the 3rd International Conference on Music Information Retrieval (ISMIR 2002)*, pages 283–284, 2002.