

Evaluating Melodic Similarity Using Pairwise Sequence Alignments

and

Suffix Trees

by

David D. Wickland

A Thesis

presented to

The University of Guelph

In partial fulfilment of requirements

for the degree of

Master of Science

in

Computer Science

Guelph, Ontario, Canada

©David D. Wickland, September, 2017

ABSTRACT

Evaluating Melodic Similarity Using Pairwise Sequence Alignments and Suffix Trees

David D. Wickland
University of Guelph, 2017

Advisor:
David A. Calvert

The music industry was the first mass-media industry to be transformed by the transition to digital. Today, music consumers have access to personal libraries, subscription-based libraries, and online content via web-media hosts or channels. Now more than ever, there are tremendous quantities of music available to the individual on a host of devices and platforms. With the explosion in access, there comes increasing demand to describe, index, retrieve, and interact with digital music. Consequently, there is increasing demand for novel and efficient ways to compare and manipulate digital music.

The goal of investigating melodic similarity is to develop ways of comparing melodies based on structural elements or patterns present in the melody. This approach differs from other music similarity measures, which typically employ low-level features, or clustering based on associations.

With robust and efficient ways of evaluating melodic similarities, we can develop new approaches to interacting with digital music.

To my father, and to anyone living with Primary Progressive Aphasia,

Parkinson's Plus, or any other form of dementia.

May we always remember where your light has shone, and all that it affirms.

ACKNOWLEDGEMENTS

A great many thanks are owed to those few people who were with me along this journey, and whose support made the realization of this work possible. To my supervisor, David Calvert, thank you for all your mentorship and patience, and for encouraging me to think about and to explore these ideas from different perspectives. You have been a constant support throughout not only this thesis, but my entire tenure, and for that I am tremendously grateful. To Jim Harley, thank you for convincing me to return to my alma mater, and for making a home for this research across departments and colleges. When we met, nearly a decade ago, you taught me my most favoured courses and inspired in me the aspirations that would one day become this work. To Fei Song, thank you for your friendship and guidance; I have really enjoyed working alongside you and learning from you in new and challenging ways, in equal measure. To my family, thank you for your love and encouragement; I cannot imagine my life without the opportunities you have afforded me. And finally to my Emily, thank you for always reminding me of my potential and my worth, thank you for your laughter and all that it sustains, and thank you for the dreams we get to share, to look forward to, and to celebrate.

Table of Contents

List of Tables	x
List of Figures	xi
1 Introduction	1
1.1 Problem Statement	2
1.2 Proposed Work	3
1.3 Thesis Statement	4
1.4 Thesis Organization	4
2 Literature Review	5
2.1 Symbolic Music Representations	5
2.1.1 Modern Western Notation	6
2.1.2 Letter & Integer Notation	7
2.1.3 Duration or Percussion	8
2.1.4 Parsons Code	8
2.2 Music Similarity	9
2.2.1 Forensic Musicology & Substantial Similarity	10
2.2.2 Audio-based Similarity	11
Temporal Features	11
Spectral Features	12
Cepstral-based Similarity	13
Summary	15
2.3 Melodic Similarity	16
2.3.1 String Reductions & Edit Distance	17
2.3.2 Markov Chains	18
2.3.3 Geometric Measurements	19
2.3.4 Vector Calculations	20
2.3.5 Melodic Contours	22
2.4 Longest Common Substring	23

2.4.1	Dynamic Programming	23
2.4.2	Suffix Trees	25
	Background	25
	Suffix Tries	26
	Compaction	28
	Implicit vs. Explicit	29
	Naïve Implementation	29
	Edge-Label Compression	29
	Suffix Links	31
	Generalized Suffix Trees	31
	Time and Space Constraints	33
2.4.3	Match Statistics	33
2.4.4	Suffix Arrays	34
2.5	Pairwise Sequence Alignment	35
2.5.1	Global Alignment	35
	Needleman-Wunsch	36
	Algorithm	36
	Back Tracing	38
2.5.2	Local Alignment	39
	Smith-Waterman	39
	Algorithm	40
	Back Tracing	41
2.6	Suffix Trees and Melodic Similarity	42
2.6.1	Indexing or Database Lookup	42
2.6.2	Melodic or Thematic Analysis	44
2.7	Pairwise Sequence Alignments & Melodic Similarity	45
2.7.1	Global Alignment	45
2.7.2	Local Alignment	46
3	Methodology	48
3.1	The Lakh MIDI Dataset	48
3.2	Data Preprocessing	49
3.2.1	Extracting Melodies	51
3.2.2	Covers or Renditions	51
3.2.3	De-duplication	52
3.2.4	Ground Truth Data	52
3.2.5	Key-Finding Algorithm	53
3.2.6	Transposition & Quantization	54
3.2.7	Encoding Melodies to an Alphabet	56
	Parsons	56
	Interval	56
	Pitch Class	57

	Duration	57
	Pitch Class with Duration	58
3.3	Assembled Datasets	59
3.3.1	Unique Songs	59
3.3.2	Cover Song Identification: Pop	60
3.3.3	Cover Song Identification: Jazz	60
3.4	Suffix Trees	61
3.4.1	Building the Suffix Tree for LCS	61
3.5	Global Alignment	63
3.5.1	Experimental Setup	64
3.6	Local Alignment	65
3.6.1	Experimental Setup	65
3.7	Cover Song Identification	66
3.7.1	Binary Classification	66
	Confusion Matrix	67
	Precision and Recall	67
	F_1 Score	68
4	Results	69
4.1	Unique Melodies	70
4.1.1	Exact Matching: Longest Common Substring	70
4.1.2	Inexact Matching	71
	Global Alignment	72
	Local Alignment	73
4.2	Cover Song Identification: Pop Music	76
4.2.1	Exact Matching	76
4.2.2	Inexact Matching	79
	Global Alignment	79
	Local Alignment	82
4.3	Cover Song Identification: Jazz Standard	89
4.3.1	Exact and Inexact Matching	90
4.3.2	Local Alignment with Minimum Length Threshold	91
4.4	Jazz Cover Song Identification with Varying Occurrence	92
5	Conclusions	96
5.1	Unique Songs	96
5.2	Cover Song Identification: Pop	98
5.3	Cover Song Identification: Jazz	100
5.4	Contributions	102
5.5	Future Work	102
5.5.1	Binary Classification	102
5.5.2	Exact Matching	104

5.5.3 Inexact Matching	104
Appendix A	106
Appendix B	114
References	119

List of Tables

2.1	Dynamic programming solution for inputs strings “ABAB” and “BABA”	24
2.2	Suffix array indexing of input string $T\$$	34
2.3	Needleman-Wunsch Dynamic Programming Table Initialization	37
2.4	Needleman-Wunsch Dynamic Programming Table Solved for Global Alignment	38
2.5	Smith-Waterman Dynamic Programming Table Initialization	40
2.6	Smith-Waterman Dynamic Programming Table Solved for Local Alignment	41
3.1	Krumhansl-Schmuckler key-finding major key profile	54
3.2	Krumhansl-Schmuckler key-finding minor key profile	54
3.3	Confusion matrix for binary classification	67
4.1	Mean, median, and mode of local alignment lengths for the various encodings	83
4.2	p -values of Wilcoxon signed-rank tests for various melodic representations in jazz cover song identification	90
4.3	p -values of Wilcoxon signed-rank tests for inexact matching using local alignment and applying different minimum alignment length thresholds	91

List of Figures

2.1	Segment of <i>Ode to Joy</i>	6
2.2	Integer notation & modern western notation	7
2.3	Duration transcription of <i>Ode to Joy</i>	8
2.4	Parsons Code transcription of <i>Ode to Joy</i>	9
2.5	Plot of the mel scale vs. frequency in Hz	14
2.6	Suffix Trie of “banana”	27
2.7	Compacted Suffix Trie of “banana\$”	28
2.8	Suffix Tree of “banana\$” Using Edge-Label Compression	30
2.9	Suffix Tree of “banana\$” with Suffix Links	31
2.10	Generalized Suffix Tree of <i>ABAB\$0BABA\$1</i>	32
3.1	An example of MIDI messages with both note_on and note_off events	50
3.2	An example of MIDI messages with only note_on events	50
3.3	Encoding of quantized note durations to alphabet Σ	58
3.4	Encoding of note value mod octave with quantized note durations to alphabet Σ	59
3.5	String concatenation to build a suffix tree from two query strings . .	62
3.6	Suffix Tree of <i>BONANZA#BANANA\$</i> to solve for the LCS	63
4.1	Distributions of exact match lengths of LCS for various melodic repre- sentations in Lakh Unique Songs dataset	71
4.2	Distributions of global alignment edit ratios for various melodic repre- sentations in Lakh Unique Songs dataset	72
4.3	Distributions of global alignment edits for various melodic representa- tions in Lakh Unique Songs dataset	73
4.4	\log_2 distributions of local alignment edit ratios for various melodic representations in Lakh Unique Songs dataset	74
4.5	\log_2 distributions of local alignment lengths for various melodic rep- resentations in Lakh Unique Songs dataset	75

4.6	Log ₂ distributions of local alignment edits for various melodic representations in Lakh Unique Songs dataset	75
4.7	Precision scores of exact matching for various melodic representations in cover song identification on Lakh Covers dataset	77
4.8	Recall scores of exact matching for various melodic representations in cover song identification on Lakh Covers dataset	78
4.9	F_1 scores of exact matching for various melodic representations in cover song identification on Lakh Covers dataset	79
4.10	Precision scores for global alignment of various melodic representations in cover song identification on Lakh Covers dataset	80
4.11	Recall scores for global alignment of various melodic representations in cover song identification on Lakh Covers dataset	81
4.12	F_1 scores for global alignment of various melodic representations in cover song identification on Lakh Covers dataset	82
4.13	Precision scores of local alignment using Parsons encoding for various alignment lengths in cover song identification on Lakh Covers dataset	84
4.14	Precision scores of local alignment using PC+Duration encoding for various alignment lengths in cover song identification on Lakh Covers dataset	85
4.15	Recall scores of local alignment using Parsons encoding for various alignment lengths in cover song identification on Lakh Covers dataset	86
4.16	Recall scores of local alignment using PC+Duration encoding for various alignment lengths in cover song identification on Lakh Covers dataset	87
4.17	F_1 scores of local alignment using Parsons encoding for various alignment lengths in cover song identification on Lakh Covers dataset . . .	88
4.18	F_1 scores of local alignment using PC+Duration encoding for various alignment lengths in cover song identification on Lakh Covers dataset	89
4.19	F_1 scores of exact matching using Parsons encoding with different occurrences of covers in the jazz Standard dataset	93
4.20	F_1 scores of global alignment using Parsons encoding with different occurrences of covers in the jazz Standard dataset	94
4.21	F_1 scores of local alignment using Parsons encoding with different occurrences of covers in the jazz Standard dataset	95
A.1	Precision scores of local alignment using Parsons encoding for various alignment lengths in cover song identification on Lakh Covers dataset	106
A.2	Precision scores of local alignment using Interval encoding for various alignment lengths in cover song identification on Lakh Covers dataset	107
A.3	Precision scores of local alignment using PitchClass encoding for various alignment lengths in cover song identification on Lakh Covers dataset	107
A.4	Precision scores of local alignment using Duration encoding for various alignment lengths in cover song identification on Lakh Covers dataset	108

A.5	Precision scores of local alignment using PC+Duration encoding for various alignment lengths in cover song identification on Lakh Covers dataset	108
A.6	Recall scores of local alignment using Parsons encoding for various alignment lengths in cover song identification on Lakh Covers dataset	109
A.7	Recall scores of local alignment using Interval encoding for various alignment lengths in cover song identification on Lakh Covers dataset	109
A.8	Recall scores of local alignment using PitchClass encoding for various alignment lengths in cover song identification on Lakh Covers dataset	110
A.9	Recall scores of local alignment using Duration encoding for various alignment lengths in cover song identification on Lakh Covers dataset	110
A.10	Recall scores of local alignment using PC+Duration encoding for various alignment lengths in cover song identification on Lakh Covers dataset	111
A.11	F_1 scores of local alignment using Parsons encoding for various alignment lengths in cover song identification on Lakh Covers dataset . . .	111
A.12	F_1 scores of local alignment using Interval encoding for various alignment lengths in cover song identification on Lakh Covers dataset . . .	112
A.13	F_1 scores of local alignment using PitchClass encoding for various alignment lengths in cover song identification on Lakh Covers dataset . . .	112
A.14	F_1 scores of local alignment using Duration encoding for various alignment lengths in cover song identification on Lakh Covers dataset . . .	113
A.15	F_1 scores of local alignment using PC+Duration encoding for various alignment lengths in cover song identification on Lakh Covers dataset	113
B.1	F_1 scores of exact matching using Parsons encoding with different occurrences of covers in the Jazz Standard dataset	114
B.2	F_1 scores of exact matching using Interval encoding with different occurrences of covers in the Jazz Standard dataset	115
B.3	F_1 scores of exact matching using Duration encoding with different occurrences of covers in the Jazz Standard dataset	115
B.4	F_1 scores of global alignment using Parsons encoding with different occurrences of covers in the Jazz Standard dataset	116
B.5	F_1 scores of global alignment using Interval encoding with different occurrences of covers in the Jazz Standard dataset	116
B.6	F_1 scores of global alignment using Duration encoding with different occurrences of covers in the Jazz Standard dataset	117
B.7	F_1 scores of local alignment using Parsons encoding with different occurrences of covers in the Jazz Standard dataset	117
B.8	F_1 scores of local alignment using Interval encoding with different occurrences of covers in the Jazz Standard dataset	118
B.9	F_1 scores of local alignment using Duration encoding with different occurrences of covers in the Jazz Standard dataset	118

Chapter 1

Introduction

There may be no limit to the depth of information that we perceive in music. Be it the style: how it might owe credence or credit to creators before, how it might diverge from past norms, or marry disparate and unlikely themes; or the sonic textures: how they inspire visions of the past, or the future, incite sensations in other senses, or alter our moods; whatever the phenomenon we experience in listening to and or creating music, we can break down all that we perceive into seemingly infinite pools of details that collectively form the obscure emotions of our experience.

Today, there is a wealth of research into understanding and categorizing many styles of music on a plethora of representational levels. Much of this work has been driven by the Musical Information Retrieval (MIR) community, and some great strides have been made in specific areas of open research. At the heart of this research, as with many other core areas of computing, is the notion of similarity. With respect to

music, there are boundless ways to describe the parallels, resemblances, divergences, or incompatibilities of two separate pieces, or even segments. The world of MIR research has produced a number of low-level features that can be extracted from musical data, and high-level representations, all of which can be used to represent various stylistic and or thematic elements of music. It is this groundwork that the research in this thesis is built upon.

1.1 Problem Statement

For a human listener, judging the similarities of two melodies at face value is perhaps intuitive or almost instinctual. While the average listener may not be able to articulate structural or relational details, they will undoubtedly be able to yield an opinion of how similar they feel the two melodies are, and this opinion will embody a wealth of subconscious understanding about the perception and internalization of music. Recreating this task on a computer proves to be incredibly challenging for a variety of reasons. Perhaps most crucial is that we do not understand how we perceive music, or more specifically, the ways we relate to its innumerable elements.

Nevertheless, computers are capable of rendering very sophisticated analyses in short order. To this end, we are able to work with the musical data features that we can extract or summarize, and evaluate them to better understand the connections that exist and how we may relate to these intrinsic properties.

The challenge of this work is to represent music melodies symbolically, and evalu-

ate various approaches to estimate similarity for not only their efficacy, but moreover their behaviour in the context of musical data.

1.2 Proposed Work

The focus of this thesis is solely on melodic similarity and does not consider other thematic or structural elements of music. For the purpose of this work, we consider various representations of transcribed melodies that preserve greater or lesser amounts of detail in the information they encode. The results obtained in this work are subject to the data available, and consequently bear some significant assumptions about the type and nature of the melodies present in the data. The results presented in this work should be interpreted within the context of their assumptions and derivations.

In our work, we seek to understand the various ways in which a melody can be represented and compared to other melodies. This work is grounded in algorithms that were historically used to sequence and analyze DNA and genome sequences. By choosing to represent melodies with an alphabet of symbols, we are able to apply the same techniques to evaluate both macro-level and micro-level sequence similarity. In particular, we evaluate global and local alignments of two music melodies at a time, using both exact and inexact matching approaches.

1.3 Thesis Statement

The purpose of this thesis is to evaluate string matching techniques as a method for determining symbolic (i.e. transcriptional) similarity of melodies. The intention is to demonstrate that employing suffix tree structures and dynamic programming can adequately differentiate unique melodies from substantially similar melodies.

1.4 Thesis Organization

The remainder of this thesis is structured as follows:

Chapter 2 describes various historical representations of music, the background of music and melodic similarity, the tools put in place in this work to evaluate similarity, and notable contributions to music analysis using these tools

Chapter 3 outlines the dataset, all of the processing necessary to perform the tests, and the measures we employ to summarize the performance of the music similarity tests

Chapter 4 demonstrates the findings from our various tests involving a variety of applications, datasets, and contexts for music similarity judgements

Chapter 5 summarizes the findings of Chapter 4 and discusses possible improvements and areas of potential for future work.

Chapter 2

Literature Review

2.1 Symbolic Music Representations

There are many symbolic forms of musical notation used to visually represent and record music. Perhaps most familiar to us is commonly known as modern western notation, and employs the use of modern musical symbols (i.e. staves and note heads) to form various notational styles of sheet music [1]. Throughout history and across all cultures there has been considerable variation in notational styles of music. In ancient China, inscriptions of tablature were employed to transcribe music for bells and for the *guzhen* stringed instrument [2]. In ancient Greece, the musical system of notation evolved over a half millennia, from scales of tetrachords or divisions of perfect fourths, to a system of fifteen pitch keys with the octave as the unifying (intervalic) structure [3]. In medieval Europe, the makings of what we now know as

modern western notation were beginning to take shape, owing in large part to the Catholic Church’s pursuit of ecclesiastical uniformity [4]. This saw the introduction of plainchant; but in the 16th century, the development of figured bass led to the first compositions based on chord progressions.

Today much of our musical notation has been digitized into common forms of digital transcriptions such as MIDI (Musical Instrument Digital Interface) or Musical Markup Language (MML). The work of this thesis will focus on melodies transcribed in MIDI format.

2.1.1 Modern Western Notation

Western notation depicts a melody in terms of absolute pitches and durations on one or more staves, with predefined registers and key signatures. The theme from Beethoven’s *Symphony No. 9*, “*Ode to Joy*” is shown in Figure 2.1. For the purposes of this work, western notation is the most explicit form of notation that will be utilized for melodies, and we will refer to this representation as Pitch Class + Duration or PC + Duration.



Figure 2.1: Segment of *Ode to Joy*

2.1.2 Letter & Integer Notation

Letter notation refers to representing notes irrespective of register, by their named note value (e.g. C, C \sharp , etc.). This notational system has an alphabet size of 12, ignoring redundancies (i.e. enharmonic equivalents). Letter notation can be a useful transcription of reduced information where register, duration, and interval are not considered. A related representation is the notion of pitch classes, where each note letter represents a pitch class, regardless of register. For the purposes of this work, we will refer to this representation as Pitch Class.



Figure 2.2: Integer notation & modern western notation

Integer notation is the translation of pitch or interval classes into whole numbers [5]. If we represent the note C = 0, then: C \sharp = 1, D = 2, ... , B = 11. The arithmetic modulo 12 is used to create octave equivalence, so notes of different registers are denoted by the same integer value. One advantage of this system is that it ignores the complications of enharmonic “spelling” of notes based on their diatonic function (i.e. the choice between C \sharp vs. D \flat). If it is desirable to understand the direction of

intervals, negative numbers can be assigned in the case of downward intervals. For the purposes of this work, both positive and negative intervals are considered and this representation will be referred to as Integer.

2.1.3 Duration or Percussion

For this work, the representation of transcribing only note durations and ignoring any pitch information is referred to as Duration. Figure 2.3 depicts *Ode to Joy* in Duration notation, shown on a single-line staff in neutral clef. This representation employs the same transcription as some non-pitched percussion (e.g. woodblock).



Figure 2.3: Duration transcription of *Ode to Joy*

2.1.4 Parsons Code

Parsons Code is a form of musical notation primarily used to identify pieces/works through musical motion. In parsons code, there is no representation for note value, register, interval, or duration. All intervals are represented as one of three characters: U (“Up”), D (“Down”), or R (“Repeat”). As such, the melody to *Ode to Joy* in Figure 2.4 can also be depicted as the string: RUURDDDDRUURDR.



Figure 2.4: Parsons Code transcription of *Ode to Joy*

2.2 Music Similarity

In recent years, tremendous effort and attention has been paid to the topic of music similarity and classification. This pursuit has, in large part, galvanized a field of research in digital signal analysis known as Music Information Retrieval (MIR). The field of MIR is an interdisciplinary field of research that seeks to retrieve information from music using scientific approaches in signal processing, machine learning and related fields. Some of the most common applications for this research include: recommender systems, track or instrument separation/recognition, automatic transcription (from recorded audio), automatic categorization, and music generation. The ability to extract micro and or macro features from a digital recording has not only been the subject of much academic research and investigation, but has also formed the backbone of a number of successful technology companies (i.e. Shazam, Pandora, Spotify,

etc.). While focusing on audio, semantic, or thematic features has proven quite effective for many of these applications in industry, none of these features consider the underlying structural elements of the music they analyze. In the case of evaluating similarity, these features are more aptly understood as measures of sonic (or sound) similarity between digital audio recordings. While the distinction is unimportant for the purpose of categorization or recommendation, these methods fall short of being able to estimate the melodic similarity of two recordings.

2.2.1 Forensic Musicology & Substantial Similarity

Forensic musicology emerged in the 1980s as an application of musicological principles to render judgements of how similar two pieces of music are in the context of plagiarism [6]. In copyright infringement cases in the United States, as in many countries today, the courts rely heavily on expert testimony to form evidence supporting or refuting claim(s) of infringement. This testimony varies considerably from case to case, and relies primarily on subjective processes that are not strictly analytical and are scarcely reproducible or predictable [7].

Substantial similarity is a standard used in US copyright law to determine whether a defendant has infringed upon the reproduction right of a copyright [8]. The standard establishes whether or not the alleged infringement has appropriated nontrivial amounts of the copyrighted work’s expression [9]. The standard arises in recognizing that the exclusive right to produce copies of a work would be worthless if copyright

infringement were restricted only to making complete and exact reproductions of a work [8].

2.2.2 Audio-based Similarity

There are a number of computational models of audio-based similarity metrics that rely on feature extraction from digital audio recordings. Some of the most common features focused on in MIR include: temporal features, spectral features, and Cepstral-based features.

Temporal Features

Low level temporal-based features are commonly employed to estimate music similarity. They are characterized as simple to extract and have intuitive physical interpretation. Common temporal-based features include: tempo, onset rate (OR), zero-crossing rate (ZCR), loudness, maximum amplitude, minimum energy, short-time average energy, etc [10]. Tempo, often represented in beats per minute (BPM), and OR have been applied as fundamental temporal descriptors in audio processing. Bogdanov et al. created a distance measure as a linear combination of the individual distance functions of BPM and OR, and demonstrated it could effectively classify more rhythmic styles of music [10]. Also, the amalgamated distance function protects against tempo duplication (doubled or halved tempos, which are fundamentally equivalent).

Temporal features are often combined with each other and with Cepstral-based features to bolster classification performance. Tzanetakis et al. combined temporal, spectral, and Cepstral features to create a genre classification scheme capable of performing with near-human classification accuracy. For temporal features, a Rhythm Histogram (RH), is created by extracting the temporal envelope and analyzing the peaks of the autocorrelation function to determine beat periodicities [11].

McKinney and Breebaart created temporal feature sets based on perceptual models of hearing and compared their performance to standard low level feature sets and Mel-Frequency Cepstral Coefficient (MFCC) sets [12]. Their psychoacoustic feature vector includes measures of roughness, loudness and sharpness including the modulation energy of these features across various frequency bands. Their auditory filterbank temporal envelope (AFTE) feature set measured DC envelope values and frequency envelope modulation energy across different frequency bands. The AFTE performed best for most genres, although MFCCs and standard low level features outperformed classification of classical music.

Spectral Features

Spectral features of recorded audio are obtained by converting the time-based audio signal into the frequency domain before analyzing. These features include: fundamental frequency (f_0), spectral: centroid, density, flux, roll-off, delta spectrum magnitude, etc. Spectral features have proven effective as classifiers of human speech,

musical genres, and general audio data (GAD) (i.e. non-musical audio excerpts) [13].

Logan and Salomon presented a music similarity function based on K-means clustering of spectral features [14]. They compute signatures for each audio segment and then compare the segments using Earth Mover’s Distance (EMD), which calculates the minimum amount of “work” required to transform one audio signature into the other.

Li and Ogihara combine Daubechies Wavelet Coefficient Histograms and timbral features to generate compact feature sets for similarity search and emotion detection in recorded music [15]. The distance metric for similarity comparisons was a Euclidean distance between the normalized representations of two feature vectors. Operating on a dataset of jazz and classical music, their similarity search achieved more than 86% perceived accuracy as compared to “average listener” results.

Cepstral-based Similarity

Cepstral-based similarity describes the estimation of timbre of a recorded audio segment; however, it does not consider or estimate some important timbral characteristics (i.e. attack or decay times). To compute the power cepstrum of a piece of digital audio, the signal is chopped into thousands of short (i.e. on the order of tens of milliseconds) frames and the MFCCs are computed, ignoring the temporal sequence of frames [16]. MFCCs are a representation of the short-term power spectrum of an audio signal, based on the discrete cosine transform (DCT) of the log power spectrum

of the non-linear mel scale of frequencies [17].

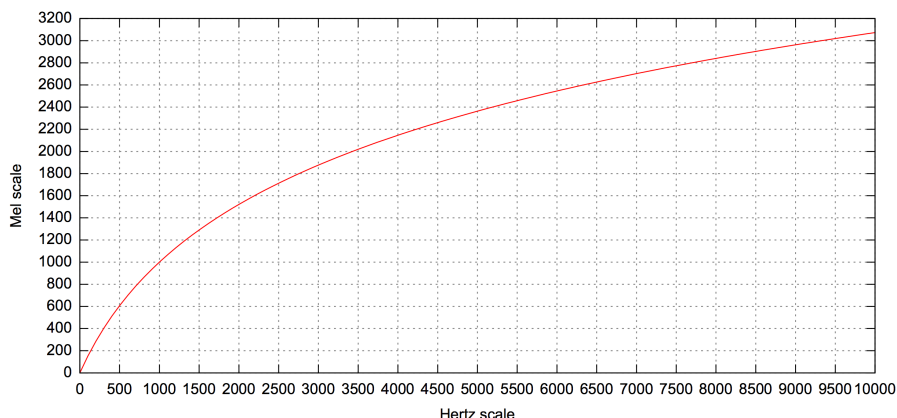


Figure 2.5: Plot of the mel scale vs. frequency in Hz

The mel scale (short for “melody”) is a perceptual scale of pitches, as judged by listeners to be equal in distance from one another [18]. Once computed, the MFCCs for the set of frames is clustered so it can be compared against cluster models of other recorded audio segments. There are many variations to computing MFCCs; however, most approaches follow some general steps with variations to the shape or size of the windows or the incorporation of dynamic features.

MFCCs are typically derived as follows:

1. Compute the Fourier transform of a windowed segment of a signal.
2. Map the spectral powers onto the mel scale with triangular overlapping windows
3. Take the logs of the powers at each mel frequency
4. Take the DCT of the list of mel log powers

5. The amplitudes of the resulting spectrum are the MFCCs.

The power cepstrum was first applied to estimate the pitch of human speech [19]. In general, MFCCs observe a spectral shape that yields information about instrumentation, production effects, the singer’s voice (if present), and timbre. MFCCs capture only a snapshot of what is typically a much longer audio recording, and consequently are not able to convey the often drastically varying qualities of different segments in a music recording. Furthermore, this representation can capture a great deal of information about the recorded audio, but it cannot interpret or convey any specific musical features.

Summary

Some of the important challenges these audio-based features face include: classification accuracy at audio segment boundaries, computational complexity, effectiveness across widely varying genres/styles of music or audio, subjectivity or ambiguity with respect to varying musical styles, to name a few. In order to improve upon these challenges, many researchers have created hybrid models of similarity or classification. While many of the aforementioned approaches have yielded impressive results, ultimately audio-based features represent measures of sound similarity and not music similarity. These audio-based features do not reflect musical similarity (in a musical sense) insofar as two vastly different recordings of an identical song would yield vastly different results for temporal, spectral and Cepstral-based feature com-

parisons. Because music recommendation and artist or song title retrieval are solved with impressive accuracy, there has been little interest on the part of industry to look at MIR approaches to more fundamental comparisons of music similarity. However, there has been some research to date focusing on this area, which is discussed in the following section.

2.3 Melodic Similarity

With the dawning of the internet age, an enormous wealth of audio and symbolic music data has become freely available to researchers and industry alike. Though melodic similarity has not been as central a focus of the MIR community as the more encapsulating field of music similarity, great strides have nevertheless been made. In order to capitalize on the significant body of online musical data, a number of algorithms and automatic tools have been developed. Melodic similarity algorithms have been developed for a host of applications including evaluating inter- and intra-work melodic relationships, evaluating underlying melodic patterns in specific genres or classes of music, and title retrieval via query by humming. In broad strokes, the melodic similarity research corpus has focused on rendering similarity judgements from one of three general perspectives: mathematical models, music theory models, and human cognitive models [20]. The remainder of this section will discuss various melodic similarity metrics and techniques.

2.3.1 String Reductions & Edit Distance

Two sequences of melodies can be represented as a string of characters (or a word) and compared in their symbolic forms using edit distance (a.k.a. Levenshtein distance). The edit distance between two strings is tabulated by counting the minimum number of operations required to transform one string into the other string [21]. Different definitions of edit distance permit different operations; however, the most typical (Levenshtein) definition considers insertions, deletions and substitutions of characters as permissible operations. Furthermore, these operations can be weighted based on their likelihood with the incorporation of a heuristic.

Edit distance can be computed in $\mathcal{O}(nm)$ time complexity with the use of well-known dynamic programming algorithms [22]. This approach has been most notably applied in the area of natural language processing (e.g. automatic spelling-correction), and in bioinformatics and computational biology (e.g. quantifying DNA sequence similarity). Nevertheless, edit distance can be criticized on the basis that while the minimum distance itself is unique, the sequence of transformations is not necessarily unique.

Miura and Shioya proposed a *pitch spectrum* measurement for melodies transcribed into a text alphabet [23]. The *pitch spectrum* is a histogram of note durations within a bar, and for each piece, every bar’s histogram is stored. Similarity is measured between a querying and candidate melody by dividing the sum of the cosine values of the histograms. The *pitch spectrum* performed well for similarity measure-

ments of variation queries as well as melody retrieval but it is presently limited to monophonic melodies.

2.3.2 Markov Chains

First-order Markov chains can be used to create a probabilistic measurement of similarity between two melodies [24]. Markov chains work by employing a square matrix M where m_{ij} is the probability that note j will follow note i . The probabilities of M are typically calculated by counting the note transitions in the melody. To calculate the overall probabilistic measurement one can either sum the logarithms of each transition probability, or take the product of the probabilities. One drawback to this approach is the need for longer sequences in order to yield reliable probabilities.

Pollastri and Simoncelli used Hidden Markov Models (HMMs) to abstract the style of five famous composers and then to classify melodies based on these trained models [25]. The HMMs achieved a classification accuracy of 42% with a limited tonal range and an order of 18; whereas humans measured 25% and 48% for amateurs and experts respectively. Hoos et al. used Markov Chains organized in a tree structure as a title retrieval system for melodic or rhythmic queries [24]. The first-order Markov chains were used to model melodic and rhythmic contours of monophonic melodies. The tree structure was ordered such that leaves represented pieces, and each internal node had a transition probability matrix.

2.3.3 Geometric Measurements

The similarity between two melodies plotted as curves can be measured in terms of the minimum area separating them. Aloupis et al. presented two algorithms to calculate the separating area: one in $\mathcal{O}(n)$ time for two melodies with fixed alignment, and the other in $\mathcal{O}(n^2 \log n)$ for two melodies with variable alignment [26]. The notion of alignment is the process of editing one melodic contour to be identical to the other melody’s contour.

Zhu and Kankanhalli presented continuous melody contours as a method for quickly estimating melodic similarity in song title retrieval for application in query by humming [27]. While this approach uses the term contours to describe changes in melodic topology, a topic later discussed in Section 2.3.5, this approach is fundamentally more geometric than those discussed later.

Melodies are constructed into a series of horizontal line segments (one for each note) from a MIDI file. This contour is scaled both horizontally and vertically such that melodies in different keys and tempos can be compared. Next, the two melodic contours are aligned horizontally and vertically by computing the centroid and shifting the querying contour to match. Melody slopes are calculated between notes in the contour that changes direction, ignoring any passing notes. The similarity of each melody slope in the querying contour is calculated using Equation 2.1.

$$S_L(k) = \frac{P(k) - \sum_i |D_S(i)|}{P(k)} \quad (2.1)$$

The pitch range of the k^{th} melody slope is denoted by $P(k)$, and the distance value of the i^{th} line segment is denoted by $D_S(i)$. Each melody slope similarity value is multiplied by a weight, W_L , shown in Equation 2.2, where $N_L(k)$ is the number of line segments in the k^{th} line slope, and N_T is the total number of line segments in the melody contour.

$$W_L(k) = \frac{N_L(k)}{N_T} \quad (2.2)$$

The overall similarity for the whole query contour is calculated using Equation 2.3.

$$S_T = \sum_{k=1}^L S_L(k) W_L(k) \quad (2.3)$$

2.3.4 Vector Calculations

Hofmann-Engl defined a measure of melodic similarity as a function of his interval vector and similarity vector [28]. The similarity vector \vec{S} , is the difference between individual pitch values of two melodic vectors M_1 and M_2 , such that:

$$\vec{S} = \{m_{21} - m_{11}, m_{22} - m_{12}, \dots, m_{2n} - m_{1n}\} \quad (2.4)$$

The interval vector \vec{I} then calculates the distances between adjacent elements of

the similarity vector.

$$\vec{I} = \{s_2 - s_1, s_3 - s_2, \dots, s_n - s_{n-1}\} \quad (2.5)$$

A transpositional similarity predictor $||\vec{F}_1||$ and interval similarity predictor $||\vec{F}_2||$ are calculated by using Equation 2.7 and 2.6 respectively.

$$||\vec{F}_1|| = \sqrt{\frac{\sum_{i=1}^n \left(e^{-\frac{k_1}{n^{C_1} s_i^2}} \right)^2}{n}} \quad (2.6)$$

$$||\vec{F}_2|| = \sqrt{\frac{\sum_{i=1}^{n-1} \left(e^{-\frac{k_2}{(n-1)^{C_2} t_i^2}} \right)^2}{n-1}} \quad (2.7)$$

The constants k_1 and k_2 are empirical and define the strength of each interval element. Similarly, c_1 and c_2 are empirical constants to define how much the length of a melody affects the similarity measure. Finally, Hofmann-Engl defines the overall similarity in Equation 2.8

$$S = ||\vec{F}_1|| \cdot ||\vec{F}_2|| \quad (2.8)$$

2.3.5 Melodic Contours

Ghias et al. explored the use of melodic contours generated from automatic pitch tracking to be used as query data in a title retrieval database for query by humming [29]. Their approach reduces the recorded (hummed) input into a parsons string representation using autocorrelation as the pitch tracking method.

Schmuckler performed a Fourier analysis of the pitch code of melodic contours to establish a measure of similarity [30]. His approach characterized contours by their relative degrees of strength of cyclic information and his experiments established that Fourier analysis could be used as a viable predictor of melodic similarity.

Jeon et al. created a signal-matching model for continuous pitch contours of the continuous dominant frequency (f_0) without reducing the contour to a string [31]. Each contour is encoded by a set of redundant wavelet coefficients that represent the shape after normalization. By directly comparing the contours instead of their string reductions, the possibility of transcriptional error is minimized. Their method allows for partial matches at arbitrary locations along the contour by employing the redundant wavelet transforms. This approach is an effective measure of contour similarity and works well in applications of title retrieval, but it relies heavily on the accuracy of the pitch-tracking method employed to yield the original contour.

2.4 Longest Common Substring

The longest common substring (LCS) problem seeks to find the longest substring common to two or more strings that constitute a set of strings. This problem differs from the longest common subsequence problem in one very particular and important way: for the longest common substring problem, all characters must occupy consecutive positions in their original sequence.

The LCS of two input strings, (L_1, L_2) , is commonly solved using either dynamic programming or a generalized suffix tree (GST). The dynamic programming algorithm costs $\mathcal{O}(nm)$ time, where $n = |L_1|$ and $m = |L_2|$, and the GST can solve the LCS in time $\mathcal{O}(n + m)$.

The generalization of LCS is known as the k -common substring problem. For a set of input strings $L = \{L_1, \dots, L_K\}$, where $n_i = |L_i|$ and $N = \sum_{i=1}^K n_i$, find the longest substring common to at least k input strings. Solving the k -common substring problem via dynamic programming takes $\mathcal{O}(NK)$ time, whereas the GST takes $\mathcal{O}(n_1 + \dots + n_k)$ time.

2.4.1 Dynamic Programming

Dynamic programming approaches the LCS problem in bottom-up fashion by trying to find the longest common suffix for all pairs of prefixes of input strings L and M

[32]. The recursive behaviour can be seen in Equation 2.9.

$$LCSuff(L_{1..p}, M_{1..q}) = \begin{cases} LCSuff(L_{1..p-1}, M_{1..q-1}) + 1, & \text{if } [S_p] = [M_q] \\ 0, & \text{otherwise} \end{cases} \quad (2.9)$$

The cells with the maximal score represent the longest common suffixes of the possible prefixes of input strings L and M ; these are labelled in red in Table 2.1. The optimal matches for the LCS problem here are “ABA” and “BAB”, each of length 3.

	0	A	B	A	B
0	0	0	0	0	0
B	0	0	1	0	1
A	0	1	0	2	0
B	0	0	2	0	3
A	0	1	0	3	0

Table 2.1: Dynamic programming solution for inputs strings “ABAB” and “BABA”

There are a handful of implementation tricks to improve memory consumption for the dynamic programming technique to LCS. In particular:

- Only hold the current and final (bottom) row in memory
- Store only non-zero table values by using hash tables instead of arrays

2.4.2 Suffix Trees

A suffix tree (also referred to as a PAT tree or a position tree) is a compacted trie containing all suffixes of a given string. The suffix trie will be discussed further in Section 2.4.2. Suffix trees allow for particularly fast string operations and have been used in a wide variety of applications. Many of the string processing applications of the suffix tree are well suited to symbolic representations, in particular melodies. As discussed in Section 2.6, suffix trees have been applied to melodies for a variety of applications including indexing and analysis. For the purpose of this work, suffix trees have been applied to melodies for their ability to solve important problems like the LCS, or longest repeated substring (LRS) quickly and efficiently. The construction of a suffix tree requires $\mathcal{O}(n)$ time and space for an input string L , where $n = |L|$. Querying for a pattern requires time $\mathcal{O}(p)$, where p denotes the length of the querying pattern [33].

Background

Weiner first introduced the concept of a suffix tree in 1973 and it was dubbed “Algorithm of the year” by Don Knuth. This original implementation read the input string from right-to-left and successively inserted suffixes on a shortest first basis. Weiner opted against the left-to-right implementation because with his current framework, extending leaf edges would have a time complexity of $\mathcal{O}(n^2)$ for each new character [34]. Instead, the right-to-left implementation permitted online construction with

considerable time savings. An online algorithm is one that is capable of processing its input in a serial (i.e. piece-by-piece) fashion such that it does not require the entire input to be available from the start.

McCreight made several improvements to the suffix tree in 1976, and changed the implementation to a left-to-right or longest suffix first. However, his method produced intermediate trees that were not suffix trees, and so the method was not online [34].

In 1995, Ukkonen presented his online version of the suffix tree that incrementally constructed the tree, character by character from left-to-right. His implementation later became known as Ukkonen's algorithm and boasts linear construction time, equal to the length of the input string. All the approaches discussed in this section perform queries with linear time for a constant sized alphabet, and have a worst-case time complexity of $\mathcal{O}(n \log n)$.

Suffix Tries

A suffix trie is a rooted tree that represents a collection of strings and has one node per common prefix [35]. For an alphabet Σ of size $|\Sigma|$, each edge is labelled with a character $c \in \Sigma$. A node can have at most one outgoing edge for a given character: $c \in \Sigma$ (i.e. each node can have $|\Sigma|$ edges, one for each character in the alphabet). Each key can be spelled out by tracing along a path starting from the root.

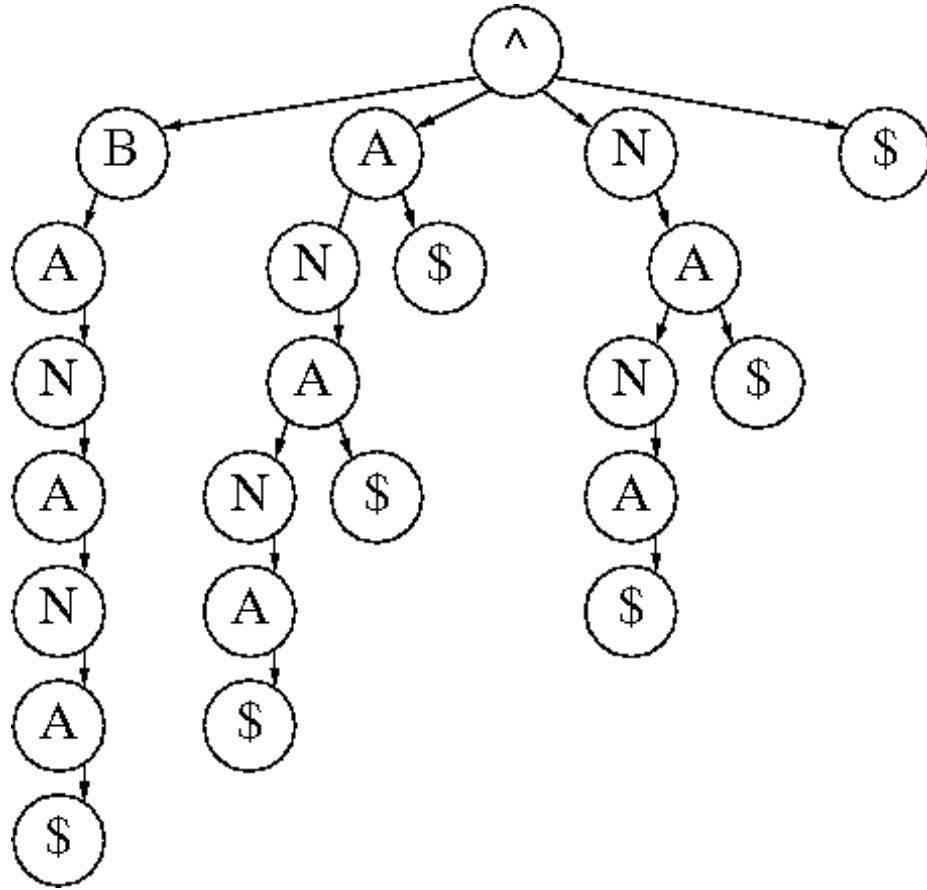


Figure 2.6: Suffix Trie of “banana”

In order to implement suffix tries, a termination character, typically denoted by \$, must be appended to the end of the input string. This termination character must be unique to the input string and must be ranked before all other characters in lexicographical order (i.e. \$ occurs before any character present in the input string). Figure 2.6 shows the suffix trie for the word “banana” where the root is denoted with ^ and the termination character is \$.

Compaction

The compaction of a suffix trie (aka compression) is performed by reducing chains of redundant nodes to a single node. In Figure 2.7, we see a compacted suffix trie where some nodes represent more than one character, as in the case of *banana\$*. A compacted suffix trie is known as a suffix tree; however, there are many further modifications to the tree structure to improve its performance and minimize its spatial cost.

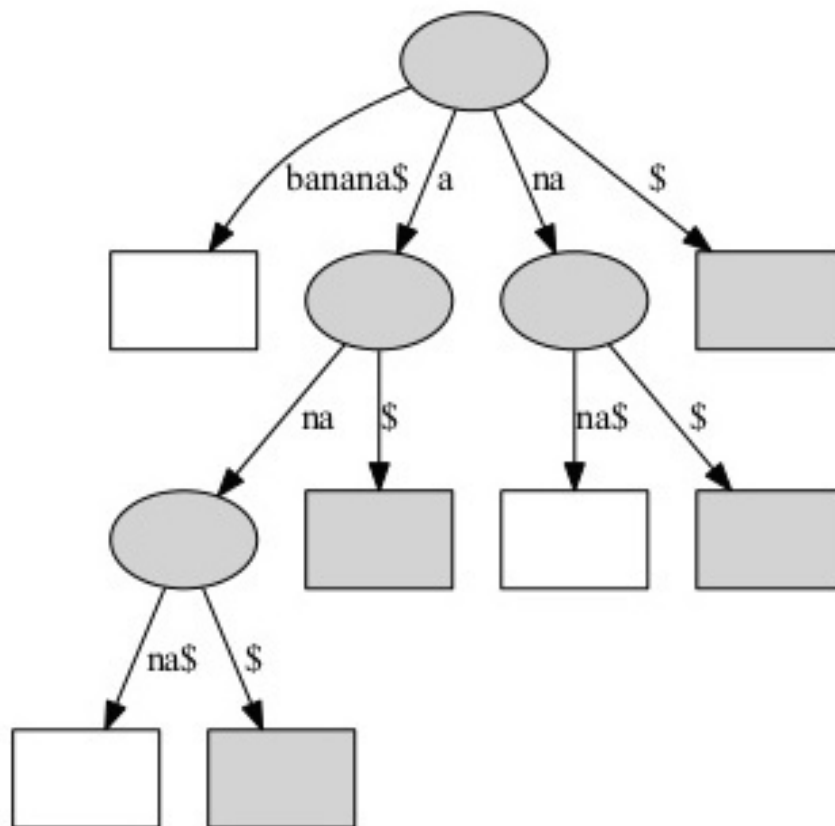


Figure 2.7: Compacted Suffix Trie of “banana\$”

Implicit vs. Explicit

A suffix tree can be either implicit or explicit. In an implicit suffix tree, suffixes end in a non-unique (non-terminating) character. Here, suffixes can end at an internal node, making them a prefix of other suffixes. In the case of *banana*, the final suffix “a” is also a prefix of “ana” and “anana”. In an explicit suffix tree, each suffix is explicit, that is to say: each suffix ends at a leaf node. This is achieved by employing the terminating character discussed in Section 2.4.2. We choose a terminating character such that it does not belong to the alphabet of the input string(s), thus ensuring this character cannot appear elsewhere in the string.

Naïve Implementation

The naïve implementation of a suffix tree requires $\mathcal{O}(n^3)$ time for construction [32]. For each phase, where phase refers to the addition of a suffix to the suffix tree, there is $\mathcal{O}(n^2)$ work, because we must update every existing node; in total there are n phases to the construction of a suffix tree. The following two sections describe implementation tricks to reduce this time complexity from $\mathcal{O}(n^3)$ down to just $\mathcal{O}(n)$.

Edge-Label Compression

Instead of using substrings as labels for nodes in the suffix tree, we can alternatively store suffixes as indexes to the input string. For example, two numbers in the two-tuple $(start, end)$ can represent the starting position and ending position of the suffix

in the original string. This reduces the size of the output from $\mathcal{O}(n^2)$ to $\mathcal{O}(n)$. In Figure 2.8 we see the use of edge-label compression where the various suffix labels of *banana*\$ along the branches are purely annotational.

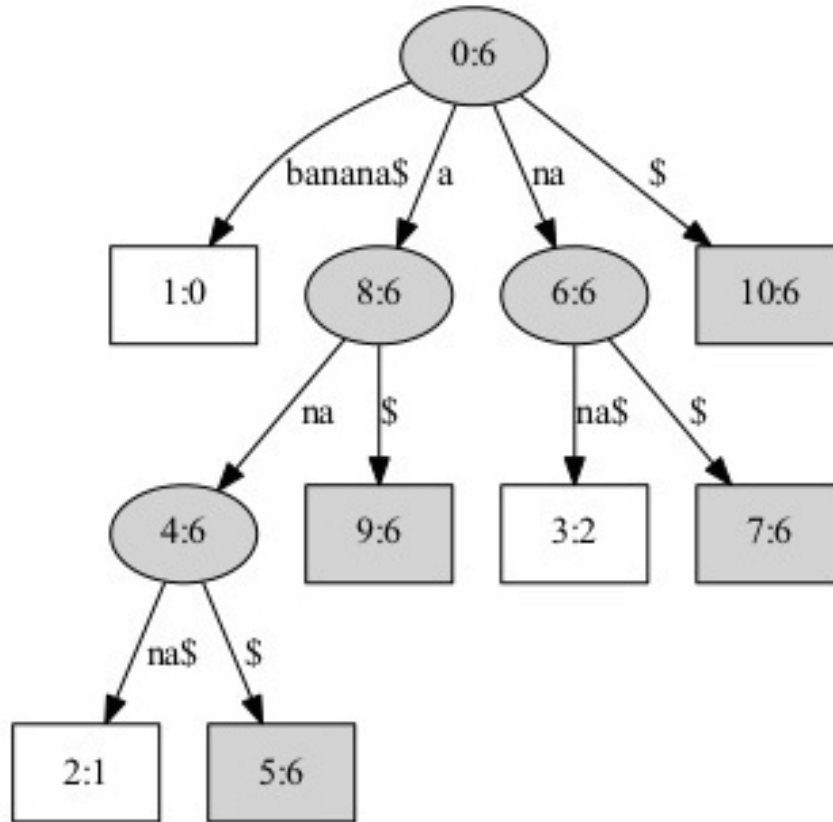


Figure 2.8: Suffix Tree of “banana\$” Using Edge-Label Compression

In Ukkonen’s implementation, the *end* label of every suffix is kept by a pointer instead of an integer, and at the beginning of each phase it is updated such that all suffixes are simultaneously updated with $\mathcal{O}(1)$ work.

Suffix Links

Suffix links are pointers from one internal node to another throughout the suffix tree. Suffix links permit extensions without the work of walking down from the root because they point to the location of the next extension. Figure 2.9 illustrates the suffix tree of *banana*\$ with suffix links present. By connecting disparate branches of the tree, the traversal work required in each extension phase is dramatically reduced.

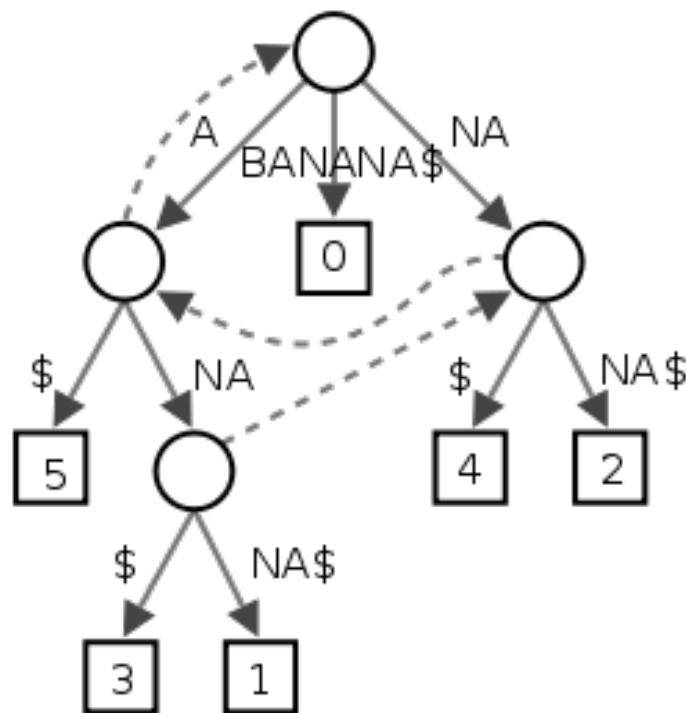


Figure 2.9: Suffix Tree of “banana\$” with Suffix Links

Generalized Suffix Trees

The GST is used in applications where more than one string needs to be searched or indexed. In order for the GST to function as desired, each input string must be

concatenated with its own unique, out-of-alphabet, termination character [36]. Then all input strings are concatenated together as the input to the GST. For example, the strings $ABAB$ and $BABA$ could be represented as follows: $ABAB\#BABAB\$$. This ensures no suffix is a substring of a substring of another suffix, and that each suffix is represented by a unique leaf node. Figure 2.10 shows the GST of this example where the input string is: $ABAB\$0BABA\1

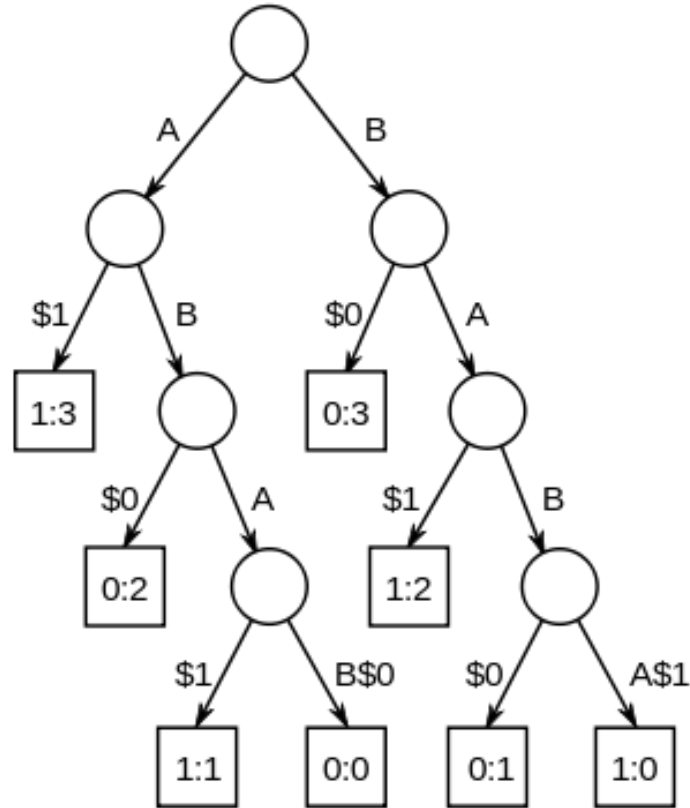


Figure 2.10: Generalized Suffix Tree of $ABAB\$0BABA\1

Time and Space Constraints

Suffix trees achieve a linear time and space bound, but have a fairly sizeable hidden constant. In some computation contexts, this is not sufficient for the desired performance. One solution to this was the development of suffix arrays. These will be discussed in detail in Section 2.4.4.

Memory constraints are common in suffix tree applications with longer inputs. For applications of more than one input string, match statistics were developed to reduce the memory requirement by only storing one of the input strings. Match statistics will be discussed in section 2.4.3.

Suffix tree performance and construction depends on alphabet size. Ukkonen's algorithm runs in linear time if the alphabet size is constant; however, for a changing alphabet size, the algorithm runs in $\mathcal{O}(n \log n)$. For the strict case, suffix tree construction is: $\min(\mathcal{O}(n \log n), \mathcal{O}(n \log |\Sigma|))$, and search times are:

$\min(\mathcal{O}(|P| \log n), \mathcal{O}(|P| \log |\Sigma|))$, where $|P|$ is the length of the querying string [33].

2.4.3 Match Statistics

Another approach to minimize space complexity for LCS and similar problems is to employ match statistics. In the case of the LCS problem, a suffix tree is created using only the shorter of the two strings. Then the second string is passed along the tree, as if adding the string to the tree; however, rather than adding the string, a table of statistics is produced to keep track of common regions. This approach can have

considerable savings in terms of spatial complexity.

2.4.4 Suffix Arrays

Suffix arrays were first introduced by Manber and Myers in 1989 and subsequently published in 1993 [37]. While both suffix trees and suffix arrays demand $\mathcal{O}(n)$ space, suffix arrays are more economical insofar as they require as few as 5 bytes per character. Recent implementations of suffix trees require upwards of 15-20 bytes per character [38]. However, suffix arrays produce a moderate increase in query time from $\mathcal{O}(m + j)$ to $\mathcal{O}(m + j + \log n)$. where m is the length of the pattern, j is the number of matches, and n is the length of the input string.

For an input string T of length n , or $T\$$ of length $n+1$, a suffix array is constructed using an array of integers from $[1..n+1]$ that specifies the lexicographic ordering of the $n+1$ suffixes of $T\$$. Table 2.2 illustrates the suffix array for an input string: *banana*. Recall, $\$$ is lexically ordered first.

i	T_i	$T_{\text{suffixarr}[i]}$	$\text{suffixarr}[i]$
1	banana\$	\$	7
2	anana\$	a\$	6
3	nana\$	ana\$	4
4	ana\$	anana\$	2
5	na\$	banana\$	1
6	a\$	na\$	5
7	\$	nana\$	3

Table 2.2: Suffix array indexing of input string $T\$$

The naïve implementation of a suffix array requires $\mathcal{O}(n^2 \log n)$ construction time

when employing an $\mathcal{O}(n \log n)$ sort algorithm [38]. Alternatively, it is possible to build a suffix tree first in $\mathcal{O}(n)$ time, then construct a suffix array from the suffix tree in $\mathcal{O}(n)$ time, and discard the suffix tree. Although, with an even greater memory requirement, this approach is not suitable to large texts/datasets. Today, many improvements have been made to suffix array construction including Burkhardt and Kärkkäinen’s algorithm which runs worst case in $\mathcal{O}(n \log n)$ time and uses $\mathcal{O}(n/\sqrt{\log n})$ space [39].

2.5 Pairwise Sequence Alignment

Pairwise Alignments are intended to compute the optimal, or best-matching, alignment (local or global) of two input sequences. The three most common techniques for producing pairwise alignments are: dot-matrix methods, dynamic programming, and word-based methods [40]. Each approach has their own benefits and drawbacks to identifying and quantifying regions of similarity between two sequences, but all approaches have difficulty with long sequences of highly repetitive (low-complexity) information.

2.5.1 Global Alignment

Global alignments attempt to align every character in every sequence using the allowable transformations, typically: insertions, deletions and substitutions, as discussed in Section 2.3.1. This approach is most useful when comparing input strings of roughly

the same size, as strings with disparate lengths will result in many insertions/deletions that can penalize the overall score of the alignment.

Needleman-Wunsch

The Needleman-Wunsch algorithm was developed by Saul B. Needleman and Christian D. Wunsch and published in 1970 [41]. It was one of the first applications of dynamic programming to evaluate biological sequence similarity and it is still commonly used today in applications of sequence alignment where the quality, or optimality, of the alignment is particularly important. The Needleman-Wunsch algorithm is a dynamic programming approach to solving the global alignment of two input strings using a substitution matrix that scores the match, mismatch, insertion, and deletion operations for an optimal alignment. The dynamic programming approach to solving the global alignment of two sequences relies on the recurrence relationship, where after the initial setup, each step (computation) towards the optimal alignment is defined as a function of its preceding terms.

Algorithm

Before describing the algorithm, a substitution matrix needs to be established to set out the score or penalty associated with a given transformation (insertion, deletion, substitution). For the purposes of this example, a match will receive a score of +1 and a mismatch, insertion, or deletion will receive a score of -1. Next, a two-dimensional

table is constructed to compare two input sequences with an additional left-most column and top-most row. This will represent the empty string (i.e. “”) as it aligns to the two sequences. Each entry in the table represents a solution to a subproblem of the global alignment, and each solution is defined in Equation 2.10

$$T_{(i,j)} = \max \begin{cases} T_{(i-1,j-1)} + \sigma_{i,j} \\ T_{(i-1,j)} + gappenalty \\ T_{(i,j-1)} + gappenalty \end{cases} \quad (2.10)$$

We initialize the upper left cell to zero, and using the above equation gives an initialized table shown in Table: 2.3. We see these descending values in the row and column associated with the empty string because every successive character requires another insertion or deletion operation in order to make the two sequences match.

	“”	B	A	N	A	N	A
“”	0	-1	-2	-3	-4	-5	-6
B	-1						
O	-2						
N	-3						
A	-4						
N	-5						
Z	-6						
A	-7						

Table 2.3: Needleman-Wunsch Dynamic Programming Table Initialization

Solving for all entries in the table, we obtain the final solution as shown in Table 2.4, where the global alignment is depicted in the path described by the red characters.

The path is solved using the back tracing algorithm described in Section 2.5.1.

	“”	B	A	N	A	N	A
“”	0	-1	-2	-3	-4	-5	-6
B	-1	1	0	-1	-2	-3	-4
O	-2	0	0	-1	-2	-3	-4
N	-3	-1	-1	1	0	-1	-2
A	-4	-2	0	0	2	1	0
N	-5	-3	-1	1	1	3	2
Z	-6	-4	-2	0	0	2	2
A	-7	-5	-3	-1	1	1	3

Table 2.4: Needleman-Wunsch Dynamic Programming Table Solved for Global Alignment

Back Tracing

The back tracing algorithm for the Needleman-Wunsch global alignment starts in the bottom right cell of the table. At each step, we choose the best score of the adjacent left cell $(i - 1, j)$, the adjacent top cell $(i, j - 1)$, and the up and left diagonal cell $(i - 1, j - 1)$. These are the same three cells that form the basis of the recurrence relationship described in Equation 2.10. If more than one cell has the maximum value for a given step, we can arbitrarily choose to prefer the diagonal. The path is traced out by repeating this process until we reach the upper left corner of the table; in this way we have described a global alignment with the minimum number of edits required to transform one sequence into the other based on the substitution matrix we employed.

2.5.2 Local Alignment

In contrast to global alignment, a local alignment can be used to identify regions of similarity within longer sequences that are broadly divergent but may contain segments of similar information. Because of this, local alignments are often more desirable; however, when working with large databases, finding the optimal local alignment through dynamic programming may not be viable due to the time requirements. Consequently, heuristic and probabilistic algorithms have been developed that do not guarantee an optimal alignment, but are capable of providing sufficiently similar matches.

Smith-Waterman

The Smith-Waterman algorithm is a variation of the Needleman-Wunsch algorithm, and was first proposed by Temple F. Smith and Michael S. Waterman in 1981 [42]. Both methods employ a substitution matrix and dynamic programming to populate a table that scores the alignment of two sequences. Unlike Needleman-Wunsch, in Smith-Waterman negative values are not allowed in the table, and so all negative values are assigned a value of zero. As a consequence, the left-most column and top-most row are initialized to zero, allowing the two sequences to be aligned in any position without penalty.

Algorithm

The Smith-Waterman algorithm generates a 2D scoring table in essentially the same fashion as Needleman-Wunsch, with the exception that negative values are not allowed, and are instead assigned a value of 0. This modification can be shown in Equation 2.11.

$$T_{(i,j)} = \max \begin{cases} T_{(i-1,j-1)} + \sigma_{i,j} \\ T_{(i-1,j)} + gappenalty \\ T_{(i,j-1)} + gappenalty \\ 0 \end{cases} \quad (2.11)$$

Using this modified function, we are able to generate an initialized table as shown in Table 2.5.

	“”	B	A	N	A	N	A
“”	0	0	0	0	0	0	0
B	0						
O	0						
N	0						
A	0						
N	0						
Z	0						
A	0						

Table 2.5: Smith-Waterman Dynamic Programming Table Initialization

For local alignments, multiple optimal alignments are possible, as in this example. Once the table is solved for all entries, we obtain the optimal alignments as shown in

Table 2.6, where one optimal alignment path is traced by the red characters and the other is traced by joining the path of the red and blue characters. The longer optimal alignment has one edit but spans more characters, and consequently both alignments receive the same score. The back tracing algorithm for Smith-Waterman is similar to the Needleman-Wunsch algorithm with a slight variation. It is described in Section 2.5.2.

	“”	B	A	N	A	N	A
“”	0	0	0	0	0	0	0
B	0	1	0	0	0	0	0
O	0	0	0	0	0	0	0
N	0	0	0	1	0	1	0
A	0	0	1	0	2	1	2
N	0	0	0	2	1	3	2
Z	0	0	0	1	1	2	2
A	0	0	1	0	2	1	3

Table 2.6: Smith-Waterman Dynamic Programming Table Solved for Local Alignment

Back Tracing

The back tracing algorithm is similar to Needleman-Wunsch but instead of starting in the bottom right corner, we begin at the cell with the highest ranking value. If there are multiple cells with the highest ranking value, this means there are multiple optimal local alignments. If we are only interested in considering one of these, we can simply choose the first maximum score we encounter as the point where trace back commences. From this position we choose the best score of the adjacent left cell $(i-1, j)$, the adjacent top cell $(i, j-1)$, and the up and left diagonal cell $(i-1, j-1)$.

This process is repeated until we reach a maximum value of zero in all three candidate cells. The zero-value cell is not included in the local alignment.

2.6 Suffix Trees and Melodic Similarity

Suffix Trees have been utilized directly for melodic similarity in a wide variety of applications. Some of these applications include: indexing or database querying, melodic or thematic analysis, and guiding machine learning processes for improvisation or musical style modelling, to name a few. This section will discuss some of the notable applications of suffix trees with melodic data available in academic literature.

2.6.1 Indexing or Database Lookup

Yu-Lung et al. implemented an explicit suffix tree where the scale position is assigned a character (i.e. “a”=“do”, “b”=“re”, “c”=“mi”, 'etc.) [43]. Once the suffix tree is built using the melody provided, a pattern sweeping step is applied to extract all repeating patterns of length > 2 . Lastly, the longest non-trivial repeating pattern is extracted. The study stops short of describing the algorithm for extracting the longest non-trivial repeating pattern, but does provide some performance analysis comparing their approach to employing an adaptive correlative matrix to solve for the same problem. The longest non-trivial repeating pattern has applications in database query for musical data, among other uses.

Ta-Chun et al. implemented a PAT tree (aka suffix tree) for database indexing and unstructured search of musical data [44]. Their method represents music objects as chords, and in turn represents chords as single characters. They make use of a B+ tree implementation of a PAT tree with a bucket size of four, which is capable of performing range searching for substrings within a given musical segment. In order to perform approximate substring matching, the tree performs n separate queries for each of the n suffixes of the input string. If none of these queries produces a match with the input string, then the query is not present in the database. While this approach is robust against input errors, there are a limited number of chordal representations supported. As well, this approach is currently not key-invariant, and consequently query strings would need to be tried against all 12 pitch classes.

Chen et al. present techniques for retrieving songs from music segments using two index structures based on augmented suffix trees [45]. Rhythmic and melodic data is extracted from audio recordings as well as melodic contours. The four melodic contours define the segment base and represent the four characters in the alphabet used to populate the suffix tree. These four contour characters are: \sqcap (type A), \sqcup (type B), \sqcap^{\perp} (type C), and \sqcup^{\perp} (type D). Each contour is represented as a 3-tuple containing the contour type, duration value, and \pm interval value. Each contour is added to the suffix tree which is further annotated with minimum and maximum values for querying duration and interval values. These values define the ranges for acceptable querying of inexact matching, that is determined by a threshold propagation function.

While this is a useful approach to managing time-efficient approximate searches in suffix trees, the space consumption can grow quickly for what is already considered a costly data structure in terms of spatial complexity. The authors suggest pruning the tree at a maximum height of 30 contours.

2.6.2 Melodic or Thematic Analysis

Jekovec et al. produced a computer aided melodic analysis tool that employs suffix trees. Intervals were added to the suffix tree as 2-tuples where the interval and duration were the first and second elements respectively [46]. The authors performed a complete traversal to evaluate all patterns based on their length, number of occurrences (i.e. frequency), and their melodic and rhythmic diversity. To evaluate the melodic and rhythmic diversity they applied Shannon’s normalized entropy and then added these scores to the product of the logarithms of the length and frequency of the pattern[46]. Those patterns that scored highest were considered most interesting.

Lartillot et al. compared a Prediction Suffix Tree (PST) against an incremental parsing algorithm for unsupervised learning of musical style to train a model on musical improvisation [47]. The PST exploited its relationship to a Probabilistic Finite Automaton in defining viable paths for the trained model to employ as melodic motifs. Their model performed a breadth-first search across all suffixes within the PST and selected melodic subsegments to perform based on three predefined probabilistic rules. Further enhancement filters were layered on top to broaden the expressiveness

of the model.

2.7 Pairwise Sequence Alignments & Melodic Similarity

2.7.1 Global Alignment

Bello developed a cover song retrieval algorithm based on approximate chord sequence matching using the Needleman-Wunsch-Sellers algorithm [48]. Major and minor triad chords were represented as characters, and the sequence was derived from audio recordings using Hidden Markov Model-based chord estimation. The sequence alignment incorporated a substitution matrix based on unitary distances for the doubly-nested circle of 5^{ths} (i.e. representing the relationships of both major and minor keys). The key-finding algorithm implemented to enable transposition for improved accuracy was a novel approach and might have benefitted from some of the existing techniques discussed in Section 3.2.5. Like in Ta-Chun et al., the limited number of chordal representations makes this approach difficult to generalize to various styles or genres of music. Interestingly, the results support the notion that applying string alignment or matching techniques to music or audio data cannot be done well without evaluating musically-meaningful segments in the data.

Ferraro and Pierre investigated both local and global alignments of monophonic melodies using various representations to evaluate melodic similarity [49]. Their

work found that contour representations were the least effective at classifying similar melodies, in part because of the limited vocabulary of the representation. They also found that employing a substitution matrix based on consonance of pitch class sets improved predictive outcomes as compared to a one-to-one penalization of insertion, deletion, and or substitution operations. As well they concluded that local alignment was a more effective method than global alignment at classifying similar melodies for their application and dataset.

There are many further applications of global alignment in melodic analysis; however, much of this work is often described as geometric measurements, as in the work discussed in Section 2.3.3.

2.7.2 Local Alignment

Frey designed a melodic similarity test based on the Smith-Waterman algorithm that employs a scoring matrix with one-to-one penalties for gaps and substitutions, and encodes note values (i.e. pitch class) confined to one octave [50]. He incorporates rhythm by chopping one bar segments of melodies into sixteenths and assigning a character for every subsegment; however, it is not clear if the repeated notes can be differentiated from the held notes over a segment greater than one sixteenth of one bar. All melodies were transposed to the key of C major; however the study did not mention if major and minor key melodies were compared in the same dataset.

Urbano et al. employed local alignment using n -grams, specifically 3-grams, and

pitch curves to evaluate symbolic music similarity [51]. Employing n -grams is the act of carving a sequence up into “grams” of length n characters or symbols, and then proceeding with the sequence analysis. Urbano et al. used a substitution matrix derived from the frequency of the n -grams in the corpus and evaluated for different representations of monophonic melodies using n -grams and pitch curves. Their work concluded that a *Shape* representation, derived by using only the first derivative coefficient at the beginning and end of the n -gram, performed the best at evaluating melodic similarity. They also suggest further work in evaluating classification performance by incorporating rank-based measure of search and retrieval tests.

Gómez et al. implemented the Smith-Waterman local alignment algorithm to query flamenco music for four different types of ornamentation [52]. Their work processed audio recordings of music into an interval based representation, and then performed the local alignment of the melody with four broad categories of embellishments to try to identify regions of the melody that would constitute ornamentation.

There is a wealth of further research in MIR and other applications of music and computation that employ pairwise sequence alignment techniques with symbolic musical data; the above is merely a brief summary of some of the most relevant work within the specific field of melodic similarity.

Chapter 3

Methodology

3.1 The Lakh MIDI Dataset

The Lakh MIDI dataset boasts a collection of 176,581 unique MIDI files of popular music. Nearly one third of these files have been matched to the Million Song Dataset, a collection of audio features and metadata for one million popular music songs compiled by Columbia University. The MIDI files were scraped from publicly-available sources on the internet and aligned to audio recordings [53]. This process was done by filtering out the vast majority of audio alignments, and then performing dynamic time warping-based methods to align the MIDI with the audio recordings. From a subset of the larger Lakh MIDI Dataset, 1111 melodies from unique popular western songs were obtained. Of these melodies, the shortest and longest melodies were 14 notes and 949 notes respectively. The process for extracting these melodies is described in

Section 3.2.

3.2 Data Preprocessing

Parsing the MIDI files was performed in Python using the publicly available Mido library. Each MIDI file in the Lakh dataset was comprised of several tracks, each with their own transcription (e.g. bass, keyboard, melody, etc.). From the Lakh subset of MIDI files, there were 2,225 that had a “melody” track associated with the file, and 1,259 of those were monophonic and unique. The time signature, ticks per beat, and all note events were recorded and transcribed into a text format along with the artist name and song title. The MIDI key signature was disregarded because it can be arbitrarily set by the user and ignored.

Note events are recorded in the MIDI format in one of two ways. The first is depicted in Figure 3.1, and employs both `note_on` events and `note_off` events. A `note_on` message is recorded with a specified channel, note value, velocity, and delta time. The delta time is the time in ticks from the previous note event, so cumulative time can be tabulated by summing all previous delta time stamps. Once a `note_on` message is recorded, the note is presumed to be on until a `note_off` message for the same note value is recorded. The duration of this note is the sum of all delta times since the `note_on` event.

```

note_on channel=3 note=74 velocity=86 time=30836
note_off channel=3 note=74 velocity=64 time=176
note_on channel=3 note=77 velocity=86 time=16
note_off channel=3 note=77 velocity=64 time=384
note_on channel=3 note=74 velocity=63 time=16
note_off channel=3 note=74 velocity=64 time=160
note_on channel=3 note=77 velocity=88 time=16
note_off channel=3 note=77 velocity=64 time=384
note_on channel=3 note=74 velocity=60 time=16
note_off channel=3 note=74 velocity=64 time=944
note_on channel=3 note=73 velocity=104 time=976
note_off channel=3 note=73 velocity=64 time=160
note_on channel=3 note=77 velocity=77 time=16
note_off channel=3 note=77 velocity=64 time=352

```

Figure 3.1: An example of MIDI messages with both note_on and note_off events

The second protocol for note events in MIDI messages is to only use note_on events and to issue a note_on event with a velocity of 0 as the corresponding note_off event for a given note. An example of this message format can be seen in Figure 3.2.

```

note_on channel=9 note=37 velocity=44 time=480
note_on channel=9 note=70 velocity=87 time=0
note_on channel=9 note=35 velocity=87 time=0
note_on channel=9 note=70 velocity=0 time=20
note_on channel=9 note=35 velocity=0 time=4
note_on channel=9 note=70 velocity=104 time=36
note_on channel=9 note=70 velocity=0 time=16
note_on channel=9 note=37 velocity=0 time=44
note_on channel=9 note=37 velocity=58 time=0
note_on channel=9 note=70 velocity=118 time=0
note_on channel=9 note=70 velocity=0 time=15
note_on channel=9 note=70 velocity=110 time=45
note_on channel=9 note=70 velocity=0 time=16
note_on channel=9 note=37 velocity=0 time=44

```

Figure 3.2: An example of MIDI messages with only note_on events

3.2.1 Extracting Melodies

Each melody track’s messages were parsed and each `note_on` event was added to a list with its note value and cumulative timestamp. If `note_off` messages were present in the track, the `note_on` event would be removed from this list when its corresponding `note_off` event was received. The duration of this note would be calculated from the timestamps and the note event would be added to a final list, denoted by its note value and duration.

In order to check against polyphonic melodies, every pair of note events throughout the sequence would be checked to see if more than 50% of their durations overlapped. If so, the melody would be disregarded altogether. In the case of no `note_off` events, the same polyphonic test is applied to ensure melodies were monophonic, and then the notes were recorded in order with their respective durations.

3.2.2 Covers or Renditions

For the purposes of this work a cover song will be defined as any rendition by the same artist or a different artist, of the same song title. There were a few song titles within the dataset that matched but were not covers, and these were identified as unique songs. There are no stipulations about how similar two songs need to be in order to be classified as a cover song, and there is no speculation about which song is the original. The method for identifying cover songs within the datasets is discussed in Section 4.2.

3.2.3 De-duplication

The original Lakh MIDI dataset was de-duplicated using the MD5 checksum of each MIDI file. While this process is quite robust, some duplicate files were able to sneak by with either different titling or track content, MIDI meta data, or with different performances of the instrumentation. From the list of extracted monophonic melodies, no more than one instance of a single song title by a given artist was kept for the unique melodies dataset. Covers were identified by duplicate versions of the same song title by the same artist with different melodic content, or the same song title by a different artist with different melodic content. The data for cover songs was removed from the unique songs dataset, described in Section 3.3.1, but kept for the analysis of cover song identification dataset, described in Section 3.3.2. Of these cover songs there were 202 renditions of 106 unique songs with cover songs in the dataset.

3.2.4 Ground Truth Data

For the cover song identification applications in this work, ground truth data was prepared in order to identify which melodic comparisons were between true covers and which were between two unique melodies. Ground truth data was assembled using the de-duplication and cover identification steps discussed above, and was validated by visual inspection. Various thresholds and tests were applied to the different algorithms and representations to evaluate their ability to correctly identify cover songs,

as discussed in Section 4.2. The ground truth data provided the data annotation necessary to evaluate classification performance.

3.2.5 Key-Finding Algorithm

Since the assigned key signature in a MIDI file is not checked against any criteria, a key-finding algorithm had to be employed to estimate the tonal center of each melody in the dataset. One of the most widely used techniques is the Krumhansl-Schmuckler key-finding algorithm [54].

The Krumhansl-Schmuckler algorithm calculates the Pearson Correlation Coefficient (PCC) for each possible major and minor key. The PCC formula is given in Equation 3.1, where \bar{x} and \bar{y} are the mean of x and y coordinates respectively. The x coordinate values are the key profile scores for a given pitch class, and the y coordinate values are the summed durations of each pitch class in the melody in question [55].

$$R = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2 \sum_{i=1}^n (y_i - \bar{y})^2}} \quad (3.1)$$

The key profiles are based on experiments by Krumhansl and Kessler, where participants were asked to rate how well a pitch class “fit” next to a prior melodic context that established a key (i.e. a cadence or scale) [56]. In this context, a higher score indicates a good fit between the pitch class and the perceived tonal center. The

major and minor key profiles are summarized in Tables 3.1 and 3.2.

do	do#	re	re#	mi	fa	fa#	so	so#	la	la#	ti
6.35	2.23	3.48	2.33	4.38	4.09	2.52	5.19	2.39	3.66	2.29	2.88

Table 3.1: Krumhansl-Schmuckler key-finding major key profile

la	la#	ti	do	do#	re	re#	mi	fa	fa#	so	so#
6.33	2.68	3.52	5.38	2.60	3.53	2.54	4.75	3.98	2.69	3.34	3.17

Table 3.2: Krumhansl-Schmuckler key-finding minor key profile

In addition to the key profiles described by Krumhansl and Kessler, the correlation coefficient can be used with other key profiles to evaluate the tonal center of a melody or piece of music. Some of these other key profiles include: Araden-Essen, Bellman-Budge, and Kostka-Payne [57]. The Krumhansl-Schmuckler key-finding algorithm, along with many other such algorithms, can suffer greatly in melodies that modulate or change tonal center for an extended duration.

3.2.6 Transposition & Quantization

In order to compare melodies on a key-invariant basis, the querying melody needed to be transposed to match the queried melody. Key signatures were assigned a value from $[0,23]$, where $[0,11]$ represented $\{C, C\#, D, \dots, B\}$ major, and $[12,23]$ represented $\{c, c\#, d, \dots, b\}$ minor. If the two melodies were mismatched major and minor, the querying melody would be transposed to the relative major or minor respectively, of

the queried melody. Equation 3.2 shows the transposition function used.

$$T = \begin{cases} ((12 - (k_2 - k_1)) \bmod 12) + 3, & \text{if } k_1 \geq 12 \text{ and } k_2 < 12 \\ ((12 - (k_2 - k_1)) \bmod 12) + 9, & \text{if } k_1 < 12 \text{ and } k_2 \geq 12 \\ (12 - (k_2 - k_1)) \bmod 12, & \text{otherwise} \end{cases} \quad (3.2)$$

Where k_1 and k_2 are the key signature values of melody 1 and melody 2 respectively; T is the value to add to all note values in melody 2 to transpose it to fit with k_1 .

Every melody's note durations were quantized down to thirty-second notes. This was done by encoding the duration into a value on the range $[0,127]$, where 0 represents a thirty-second note, and 127 represents a four-bar note in 4/4 time.

$$q_i = \left\lfloor \frac{d_i}{tpb \times met} \times 32 \right\rfloor \quad (3.3)$$

Where q_i is the quantized duration of the original duration d_i , tpb is the number of ticks per beat in the MIDI meta data, and met is the meter of the melody, also specified by the MIDI meta data. For met we only consider the numerator of a time signature as we manage tempo-duplication during the encoding of melodies, see Section 3.2.7. For this reason, the constant 32 is applied in the equation because it represents the alphabet size $|\Sigma|$ divided by the assumed denominator of the meter (i.e. $|\Sigma| \div 4 = 128 \div 4 = 32$).

3.2.7 Encoding Melodies to an Alphabet

There are five distinct encodings of varying degrees of information resolution for the melodies to be encoded into an alphabet. The alphabet size varies for each encoding based on the amount of information present in that particular encoding. In all encodings presented in this work, characters were simply represented with an integer value.

Parsons

For Parsons encoding, there are only three characters that constitute the whole alphabet: $\{U, D, R\}$ (i.e. “Up”, “Down”, “Repeat”). Because these characters represent a measure between-notes and not a value for each note, each melody length would need to be reduced by one. In the *Ode to Joy* example in Section 2.1.4, there are 15 notes but only 14 Parsons characters to represent them. To solve for the LCS problem, we need two additional delimiting characters, bringing the total alphabet size to $|\Sigma| = 5$.

Interval

For the interval encoding, the range of integers in the alphabet is: $[-127, 127]$. In practical terms, the largest single interval is considerably smaller than ± 127 ; however, the efficiency gained by scaling the alphabet down further didn’t warrant exploring to find the largest interval in the dataset. As before, the input string lengths are one less than the melody lengths since intervals represent a between-note value. Again

we have two additional delimiting characters, so $|\Sigma| = 258$. Due to the fact that negative values are reserved for suffix index labelling, discussed further in Section 3.4.1, we need to shift the alphabet up by 127 characters such that the range was $[0, 257]$, including the two delimiting characters. The output data was scaled back to the normal range, where a value of 0 would mean a repeated note.

Pitch Class

This encoding matches the letter and integer notation described in Section 2.1.2. For this and all further encodings, every note in a melody has a corresponding character in the encoding, so the length of a given melody will match the length of its input string. The integer range $[0, 11]$ represents the set of characters: $\{C, C\#, D, \dots, B\}$, and consequently there is no concept of register, interval, or note direction in this encoding. With the addition of the two delimiting characters, this alphabet has $|\Sigma| = 14$.

Duration

The Duration encoding ignores note values entirely and simply encodes the quantized duration value (i.e. note length) of every note event in the melody. Figure 3.3 shows the encoding of various note durations from a minimum of one thirty-second note up to a maximum of a note spanning four bars (i.e. a Longa or a quadruple whole note) [58]. From Figure 3.3, we can see the range of this alphabet is $[0, 127]$, for an

alphabet size of $|\Sigma| = 130$ including the two delimiting characters.

0	1	2	3	...	7	...	15	...	31	...	127
				...		...		...		...	

Figure 3.3: Encoding of quantized note durations to alphabet Σ

In order to account for tempo duplication, where one melody is similar but all durations are either double or half those of the other melody's, we run three separate tests of the Duration encoding. The first test evaluates both melodies at their current tempo; the second test rewrites the second melody at half the tempo (i.e. each note duration is doubled), and the third test rewrites the first melody at half the tempo and restores the second melody to its original tempo. From these three tests, the best result is stored for further analysis.

Pitch Class with Duration

Encoding both the pitch class and duration requires a considerably larger alphabet size. For this particular encoding there are 12 possible pitch class values, each with 128 possible durations values, giving an alphabet size of $|\Sigma| = 1538$, including the two delimiting characters. Figure 3.4 shows the encoding whereby the integer values $[0, 127]$ represent all possible note durations for the pitch class value C, and correspondingly $[128, 255]$ represents the range of note durations for C#, and so on.








0	1	2	3	...	128	...	256	...	1535
				...		...		...	

Figure 3.4: Encoding of note value mod octave with quantized note durations to alphabet Σ

As with the Duration encoding, discussed in Section 3.2.7, we again test for tempo duplication by comparing each pair of melodies in three separates tests, and storing the best result for further analysis.

3.3 Assembled Datasets

From the Lakh MIDI Dataset described in Section 3.1, a number of datasets were derived for the specific tests performed in this work.

3.3.1 Unique Songs

For the application of evaluating exact and inexact matching on unique songs, a dataset of 1,111 unique melodies was obtained from the Lakh MIDI dataset using the preprocessing described in Section 3.2. This dataset will be referred to as the Lakh Unique Songs dataset. A dataset of 1111 melodies yields 616,605 melodic comparisons. Extensive de-duplication was performed, as described in Section 3.2.3, in order to remove all occurrences of duplicate songs, renditions, or covers.

3.3.2 Cover Song Identification: Pop

For testing exact and inexact matching in the binary classification of cover song identification, a dataset of 1259 melodies was assembled from the Lakh MIDI dataset. This dataset will be referred to as the Lakh Covers dataset. De-duplication of identical melodies by the same or other artists was performed as described in Section 3.2.3; however, all covers were kept, unlike in the Lakh Unique Songs dataset. The Lakh Covers dataset contains 106 unique songs with at least one or more covers present in the dataset, for a total of 202 cover melodies and 1,057 completely unique melodies.

3.3.3 Cover Song Identification: Jazz

Cover song identification of jazz standards was performed with a variety of datasets. For the evaluation of exact and inexact matching, described in Section 4.3, two datasets were produced. The Summertime dataset was comprised of seven renditions of George Gershwin’s *Summertime*. These MIDI versions of the popular jazz standard were pulled from publicly available websites and the monophonic melodies were manually extracted. The control dataset was comprised of seven unique songs randomly selected from the Lakh Unique Songs dataset.

For the jazz cover song identification tests using binary classification of exact and inexact matching, described in Section 4.4, three separate datasets were assembled to vary the occurrence of covers within the corpus. These three datasets comprised 14 melodies at 50% occurrence, 77 melodies at 10% occurrence, and 707 melodies at 1%

occurrence, respectively. All non-cover melodies were again selected randomly from the Lakh Unique Songs dataset and duplicates were removed as necessary using the same techniques described previously in Section 3.2.3.

3.4 Suffix Trees

Suffix Trees were built for all melody evaluations and comparisons using Ukkonen’s implementation, discussed in Section 2.4.2. For each comparison, the LCS was solved and the length of the longest exact match was saved as well as a transcription of the match in the respective encoding.

3.4.1 Building the Suffix Tree for LCS

In order to solve for the LCS of two strings or melodies, a suffix tree needs to be built containing both melodies in one input string. This was done by concatenating one unique delimiting character at the end of the first melody, then appending the second melody, followed by a second unique delimiting character, all into one string. The two unique delimiting characters needed to be lexicographically smaller than all other characters present in the alphabet, Σ . Figure 3.5 shows how two strings, in this example $X = BONANZA$ and $Y = BANANA$, are appended together with the delimiting characters $\#$ and $\$$. Ukkonen’s implementation adds characters to the suffix tree in left-to-right fashion.

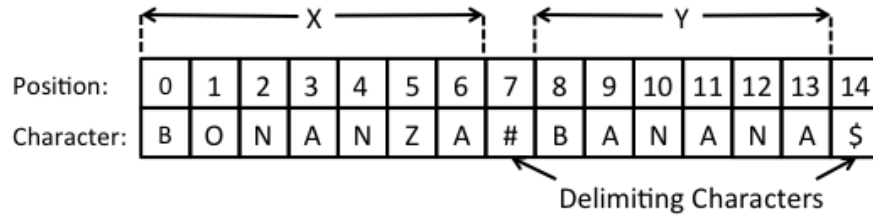


Figure 3.5: String concatenation to build a suffix tree from two query strings

Once the Suffix Tree is constructed, all the internal nodes must be annotated by doing a traversal of the tree. Each internal node should be annotated as an X node: where all children are suffixes of X , a Y node: where all children are suffixes of Y , or an XY node: where the node has suffixes of both X and Y . In this particular implementation, we assign a value of -2 to the suffix index of an X internal node, -3 to the suffix index of a Y internal node, and -4 to the suffix index of an XY internal node. During this traversal, we can keep track of the deepest XY node, which represents the LCS of the two strings. In the example shown in Figure 3.6, there are two solutions to the LCS: $ANAN$ and NAN , each of length 3.

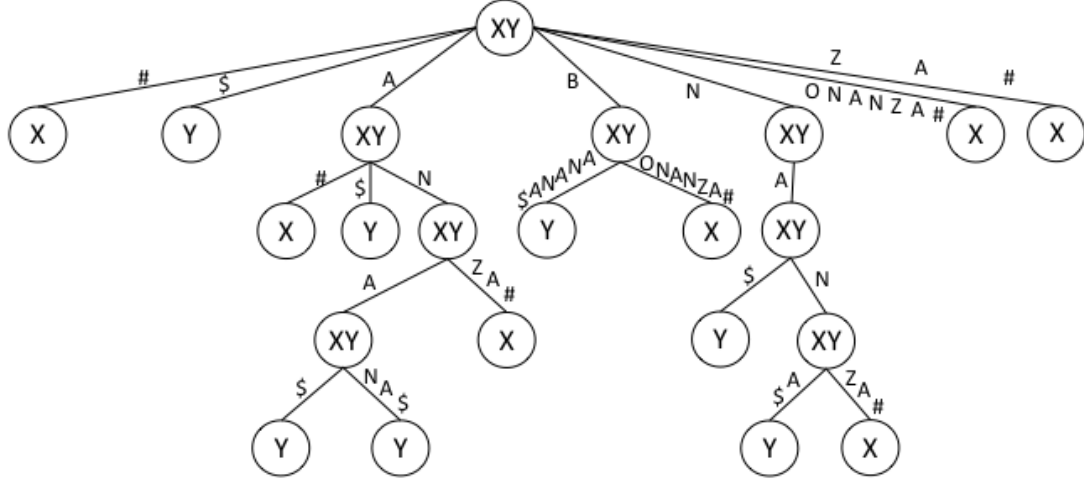


Figure 3.6: Suffix Tree of *BONANZA#BANANA\$* to solve for the LCS

Every melody in the preprocessed Lakh MIDI dataset was compared to solve for the LCS. Both the length and the LCS sequence of notes for each comparison were saved for further analysis.

3.5 Global Alignment

This section will discuss the experimental setup of the global alignment comparisons of melodies in the various datasets. The global alignment of two sequences gives the optimal alignment of two input strings based on the substitution matrix provided. For the purposes of this work, all allowable transformations were penalized equally at a cost of 1. In the case of this research, under this scoring scheme the global alignment equates to Edit or Levenshtein Distance, as discussed in Section 2.3.1.

3.5.1 Experimental Setup

Using the encoded melodies, an $(n + 1) \times (m + 1)$ sized similarity matrix was constructed in the same fashion as illustrated in Table 2.3 for each encoding. Here, n is the length of the querying melody, and m is the length of the queried melody. Once initialized, to solve for the optimal global alignment, the table was populated using the algorithm described in Section 2.5.1 and the length of the optimal alignment and number of edits were recorded from the back tracing step, discussed in Section 2.5.1. Every pair of melodies in the dataset was compared to compute the length and number of edits required for their optimal global alignment for each encoding. All datasets were processed with the same procedure regardless of size or length of melodies.

For the Duration and PC+Duration encodings, in order to account for tempo-duplication, three separate comparisons were performed and the optimal result was stored. The process of encoding a pair of melodies to mitigate tempo-duplication is discussed in Section 3.2.6. In order to rank the three possible alignments, the similarity score was calculated using Equation 3.4, where *edits* is the number of transformation operations performed, and $length_a$ and $length_b$ are the lengths of the aligned strings. Under a different scoring scheme, where the substitution matrix has different values associated with different transformations, *edits* would be the local

alignment score.

$$sim = 1 - \frac{edits}{\max(length_a, length_b)} \quad (3.4)$$

Conversely, we can describe the edit ratio or dissimilarity score as in Equation 3.5.

$$edit\ ratio = \frac{edits}{\max(length_a, length_b)} \quad (3.5)$$

3.6 Local Alignment

This section will summarize the experimental setup of the local alignment comparisons of melodies for the various datasets. The local alignment of two sequences identifies the region of greatest similarity, based on the substitution matrix. Again, all allowable transformations were penalized at a cost of 1. This provides the equivalent of an Edit or Levenshtein distance for a pair of optimally, locally-aligned substrings.

3.6.1 Experimental Setup

For each melodic encoding, an $(n + 1) \times (m + 1)$ sized table, known as the scoring matrix was generated to solve for the local alignment. Again, n is the length of the first melody, and m is the length of the second melody. All cells in the first row and column are initialized to 0 as discussed in Section 2.5.2, such that any alignment to this entry is compared to the empty string (i.e. “”). The function for calculating

successive entries in the dynamic programming table is described in Equation 2.11 and the back tracing algorithm is described in Section 2.5.2.

Tempo-duplication was again managed by calculating the similarity score with Equation 3.4. This time, $length_a$ and $length_b$ represent the lengths of the locally-aligned substrings respectively. Additionally, for evaluating the best local alignment against tempo-duplication, an additional rule was added that both $length_a > 0$ and $length_b > 0$. This avoids any reward for an alignment of length zero with zero edits.

3.7 Cover Song Identification

The cover song dataset was evaluated for melodic similarity between all 791,911 pairs of melodies using the five melodic representations discussed in Section 3.2.7. In order to evaluate the predictive power of exact and inexact matching for each representation, a binary classification scheme was setup to distinguish cover song melodies from unique melodies.

3.7.1 Binary Classification

Binary classification is the process of classifying elements in a given set into two groups or subsets on the basis of one or more classification rule(s). In order to evaluate the performance of a system as a binary classifier, we require annotated (i.e. “ground truth”) data. For the task of cover song identification, this data was assembled from

the 1,259 unique melodies. Two melodies were identified as covers if their song titles matched but their melodic sequences were unique. Songs with the same title but a different artist name were checked individually to determine if they were unique songs or if they were covers.

Confusion Matrix

The confusion matrix, shown in Table 3.3, is a contingency table that illustrates the relationship between the four possible states in a binary classification scheme. Here, tp , tn , fp , and fn represent the proportion of true positive, true negative, false positive, and false negative instances respectively. In statistics, fp and fn are often referred to as Type 1 and Type 2 error.

	Predicted Yes	Predicted No
Actual Yes:	tp	fn
Actual No:	fp	tn

Table 3.3: Confusion matrix for binary classification

Precision and Recall

Precision is the proportion of true positive or relevant instances among the predicted or retrieved instances in a dataset [59]. Recall is the proportion of true positive or relevant instances among the total positive or relevant instances in a dataset [59].

These two measures of relevance are better understood in Equations 3.6 and 3.7.

$$\text{precision} = \frac{tp}{tp + fp} \quad (3.6)$$

$$\text{recall} = \frac{tp}{tp + fn} \quad (3.7)$$

F_1 Score

In binary classification and statistics, the F_1 score measures predictive accuracy by evaluating both the precision and recall of a test or classification scheme [60]. The traditional F_1 score calculates the balanced harmonic mean of the precision and recall scores of the system, as shown in Equation 3.8

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \quad (3.8)$$

Chapter 4

Results

This section provides a summary of the testing and results for various melodic comparisons of exact and inexact matching in a variety of contexts and datasets. The Lakh Unique Songs dataset was used to evaluate the system using only melodies from unique songs, and contains no covers, renditions, or duplicates. The results from the analysis of melodic similarity for unique melodies on a moderately large corpus will help to inform the results discussed in later sections. These results will also provide a baseline of the amount of types of similarity prevalent in the datasets. The Lakh Covers dataset contains unique melodies with some covers, and was used to evaluate the performance of exact and inexact matching for cover song identification. This dataset provides an example of cover song identification with the pop music genre, where melodies from different renditions do not vary as greatly as in some other styles of music. Lastly, the Summertime dataset was used to evaluate the performance of

exact and inexact matching for cover song identification in a different genre, namely jazz, where the types and occurrences of melodic and rhythmic variations were much greater. The size of the Summertime dataset is quite small, due in large part to the availability of unique renditions of the jazz standard; however for the purposes of cover song identification a few datasets with varying occurrences of the covers have been synthesized in order to better understand the effects of diminishing probability on the application of cover song identification.

4.1 Unique Melodies

4.1.1 Exact Matching: Longest Common Substring

Figure 4.1 shows the exact match length distributions of the various encodings. These results were obtained by solving the LCS of each melodic comparison in the Lakh unique songs dataset and counting the lengths of the longest exact matches. We expect to see encodings with a smaller alphabet size and less information will result in longer matches. Figure 4.1 illustrates that the Interval, PitchClass and Duration encodings, while all representing melodies quite differently, produce effectively the same match length profile across this dataset. This perhaps suggests that there is approximately as much rhythmic variation as melodic variation within the melodies in this dataset; as well, there may be a very negligible effect to representing melodies with PitchClass notation as opposed to Interval. Parsons encoding maintains the

least amount of information and consequently produces the broadest distribution with longest match lengths, while PC+Duration encodes the most information and peaks rather narrowly, though considerably higher, with shorter match lengths than the other encodings.

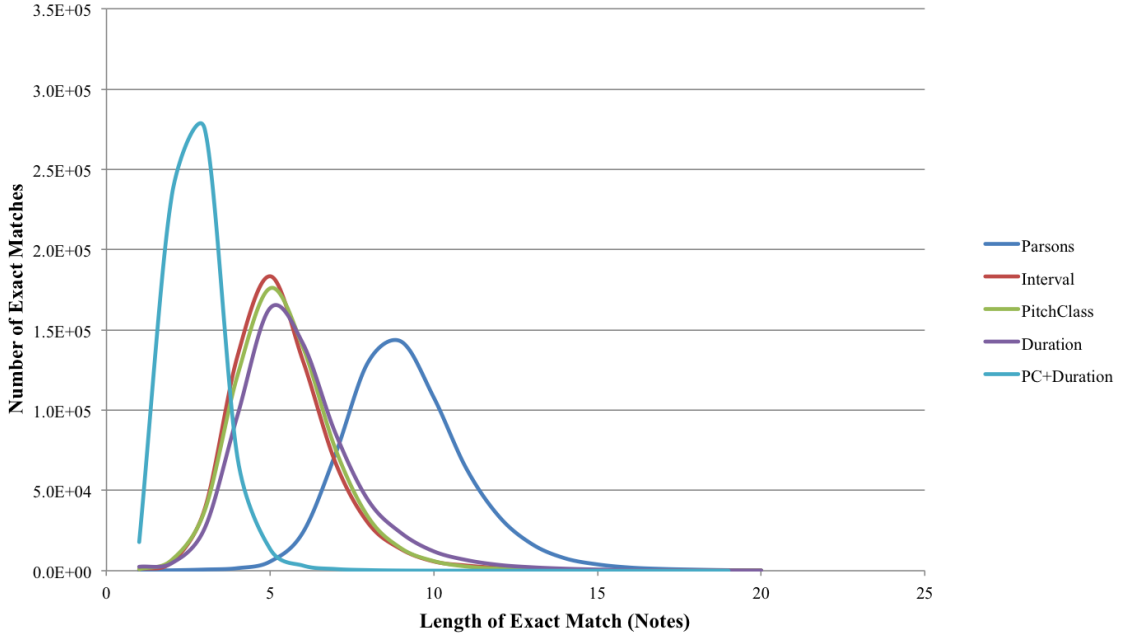


Figure 4.1: Distributions of exact match lengths of LCS for various melodic representations in Lakh Unique Songs dataset

4.1.2 Inexact Matching

For inexact match comparisons, both global and local alignments were evaluated. Both the global and local alignment scores were counted based on the edit ratio, described in Equation 3.5.

Global Alignment

The distribution of the edit ratios for all melody comparisons in the dataset is shown in Figure 4.2. Parsons encoding, being the most general, demonstrates the highest overall similarity (lowest dissimilarity) between all melodies, whereas for the PC+Duration encoding, all melodies require the greatest overall number of edits to transform one into the other.

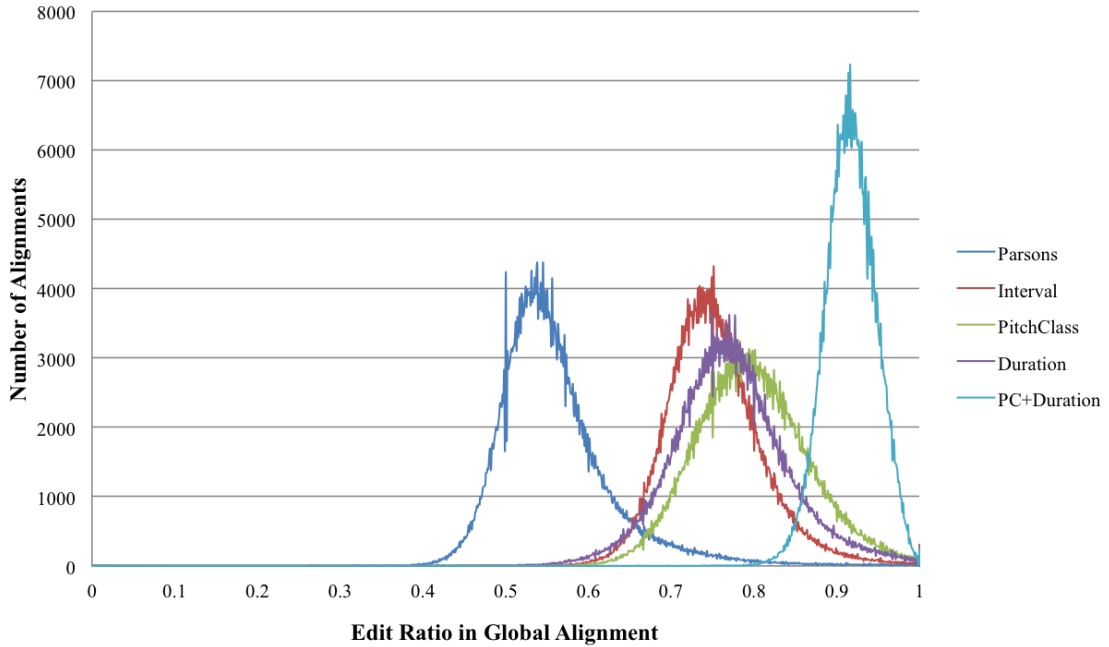


Figure 4.2: Distributions of global alignment edit ratios for various melodic representations in Lakh Unique Songs dataset

In Figure 4.3, we depict the distribution of the number of edits for each encoding to perform a global alignment between two melodies. In this case, Parsons peaks narrowly and at a value of fewer overall edits as compared to the other encodings. The other encodings, by contrast, peak much more broadly, at lower values, and with

greater overall edits required for the alignments. PC+Duration requires the most edits of any encoding; although, the difference in the PC+Duration distribution to the Interval, PitchClass and Duration encodings is modest. There is no need to show the distribution of match lengths for global alignment as that is simply a reflection of counting the lengths of the various melodies in the dataset.

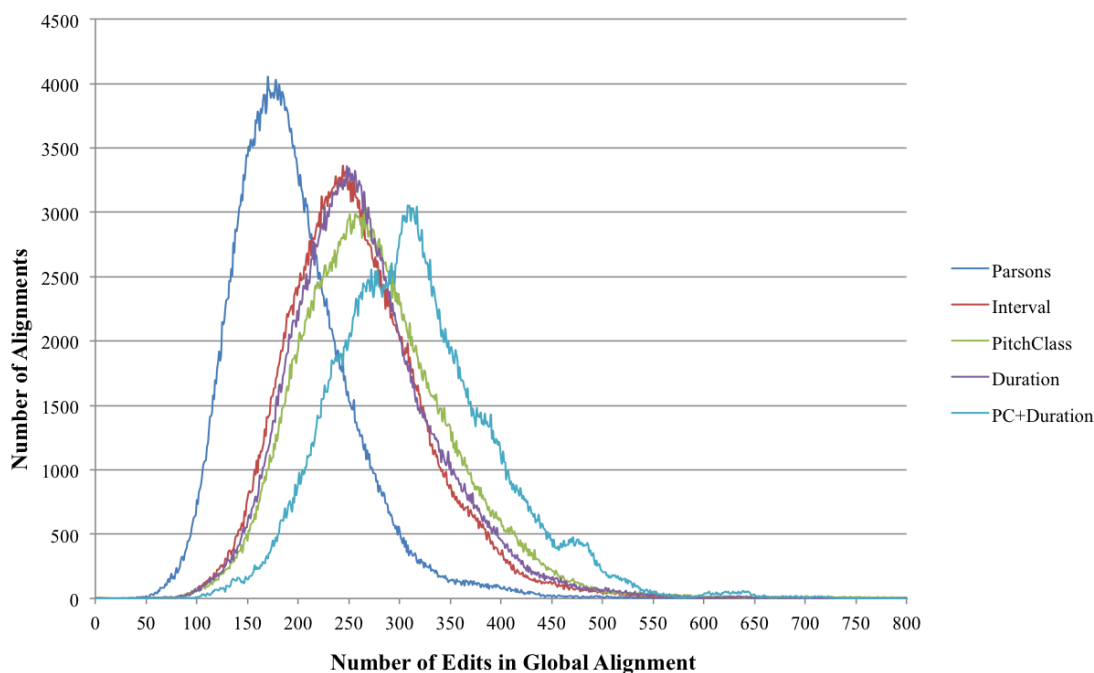


Figure 4.3: Distributions of global alignment edits for various melodic representations in Lakh Unique Songs dataset

Local Alignment

For local alignment, the distribution of edit ratios is shown in Figure 4.4. Here there is a large emphasis on zero edit local matches due to the scoring behaviour of the Smith-Waterman algorithm, which favours shorter regions of greater similarity. Again

we see Parsons as the most general encoding and PC+Duration as the least. However, in local alignments, the Duration encoding appears to have greater specificity than in global alignments or exact matching.

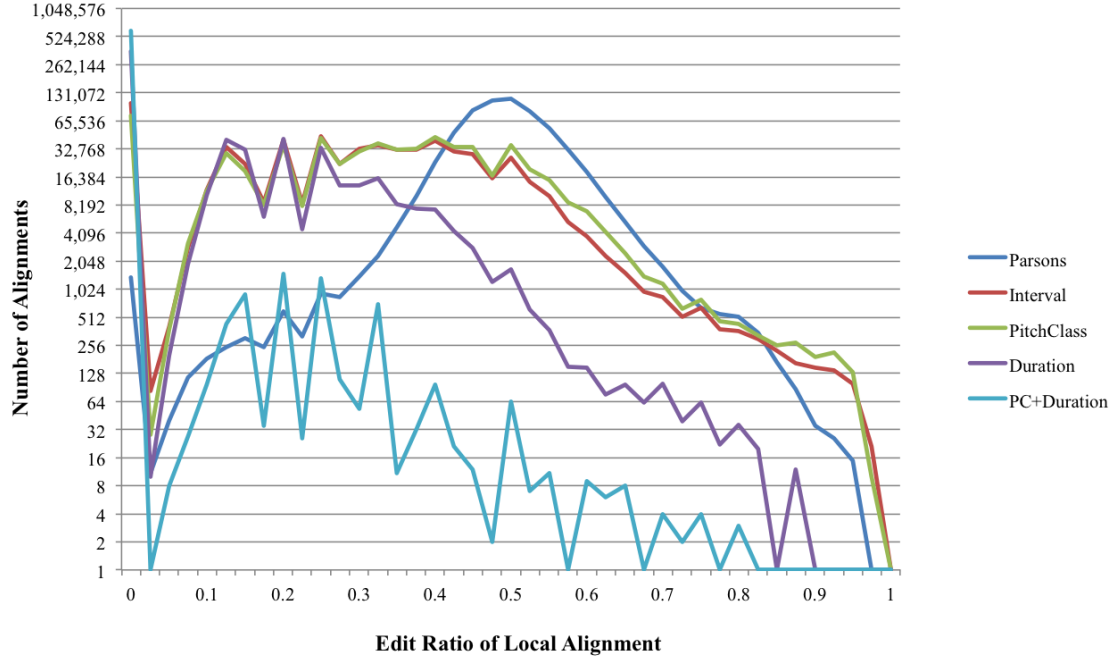


Figure 4.4: \log_2 distributions of local alignment edit ratios for various melodic representations in Lakh Unique Songs dataset

We can observe the emphasis on matching shorter regions of greater similarity in Figures 4.5 and 4.6, where we see the same emphasis towards both the shorter match lengths and fewer edits respectively. This trend hastens towards zero length matches (i.e. alignments of length zero) in both cases as the specificity of the encoding increases.

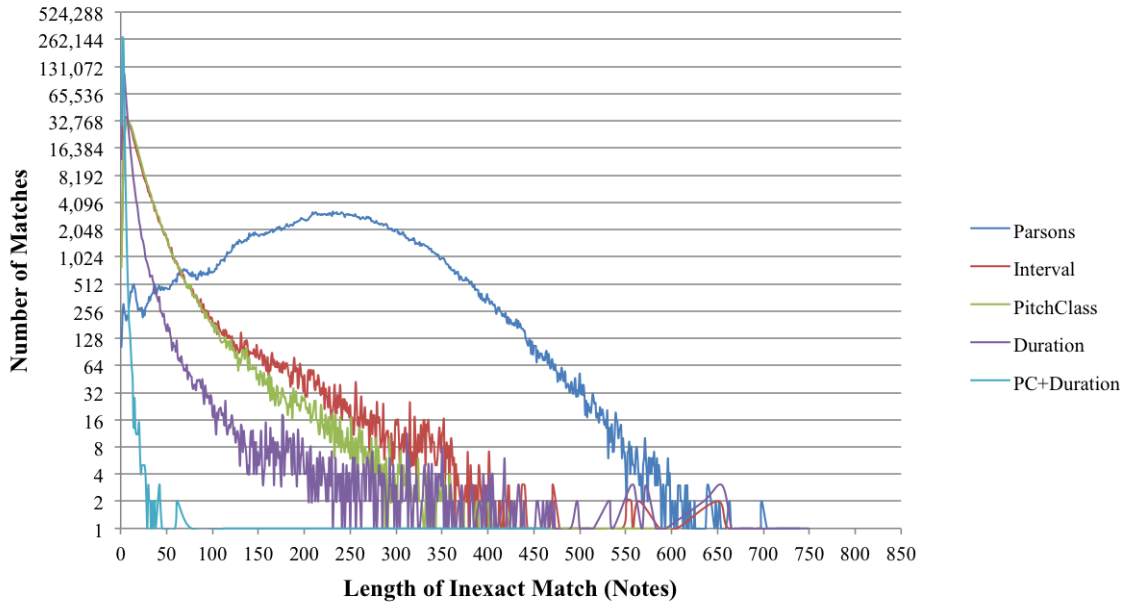


Figure 4.5: Log₂ distributions of local alignment lengths for various melodic representations in Lakh Unique Songs dataset

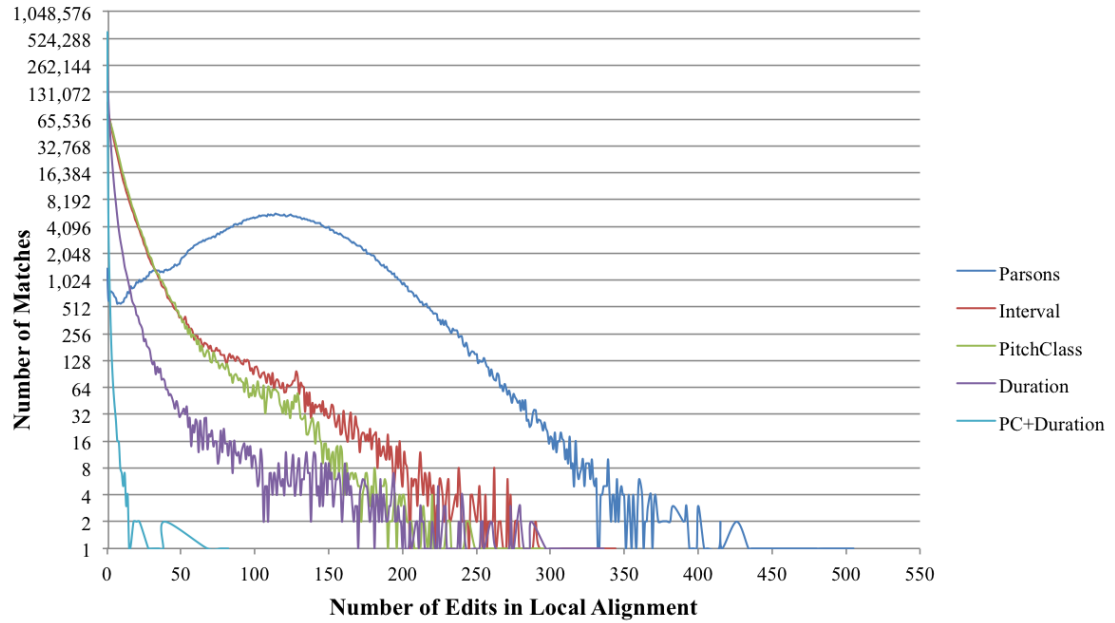


Figure 4.6: Log₂ distributions of local alignment edits for various melodic representations in Lakh Unique Songs dataset

4.2 Cover Song Identification: Pop Music

In this section, we discuss the results of using the Lakh Covers dataset with annotated cover songs to evaluate the performance of exact and inexact matching in cover song identification. The dataset, described in Section 3.1 has 1259 melodies, of which there are 202 renditions of 106 unique melodies; all other melodies in the dataset do not have an accompanying cover or rendition. With this dataset, we have a total of 791,911 melodic comparisons, of which 236 will be true covers, or an occurrence of covers of 0.03%. The intention of this section is to evaluate the performance of exact and inexact matching approaches to identify cover songs within a dataset of both unique melodies and renditions or covers.

Ground truth data was created to evaluate the predictive performance of the various testing environments of exact and inexact matching. In order to evaluate the performance of this system across all melodic encodings, a binary classifier was employed, as described in Section 3.7.1.

4.2.1 Exact Matching

The precision scores of each melodic encoding for various exact match lengths are displayed in Figure 4.7. We can see that all encodings achieve high precision scores; however, the greater specificity of the PC+Duration encoding allows this representation to reach such high precision scores with shorter match lengths than the other encodings.

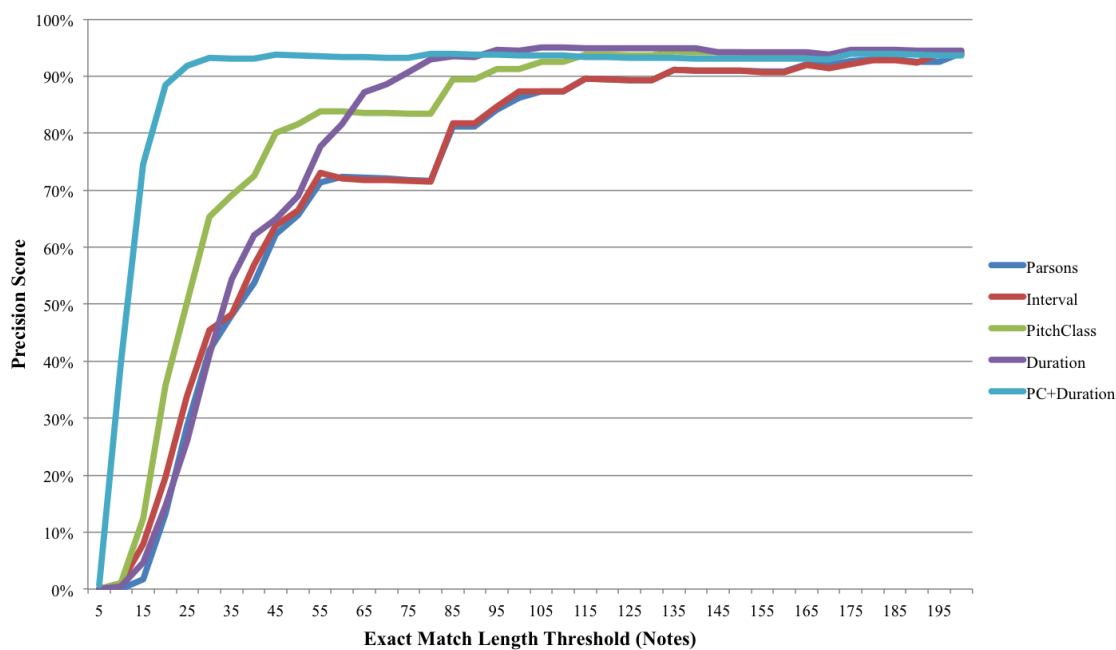


Figure 4.7: Precision scores of exact matching for various melodic representations in cover song identification on Lakh Covers dataset

The recall scores of each encoding are plotted against exact match lengths in Figure 4.8. Here we can see that all encodings achieve modest recall scores but PC+Duration struggles as compared to the other encodings.

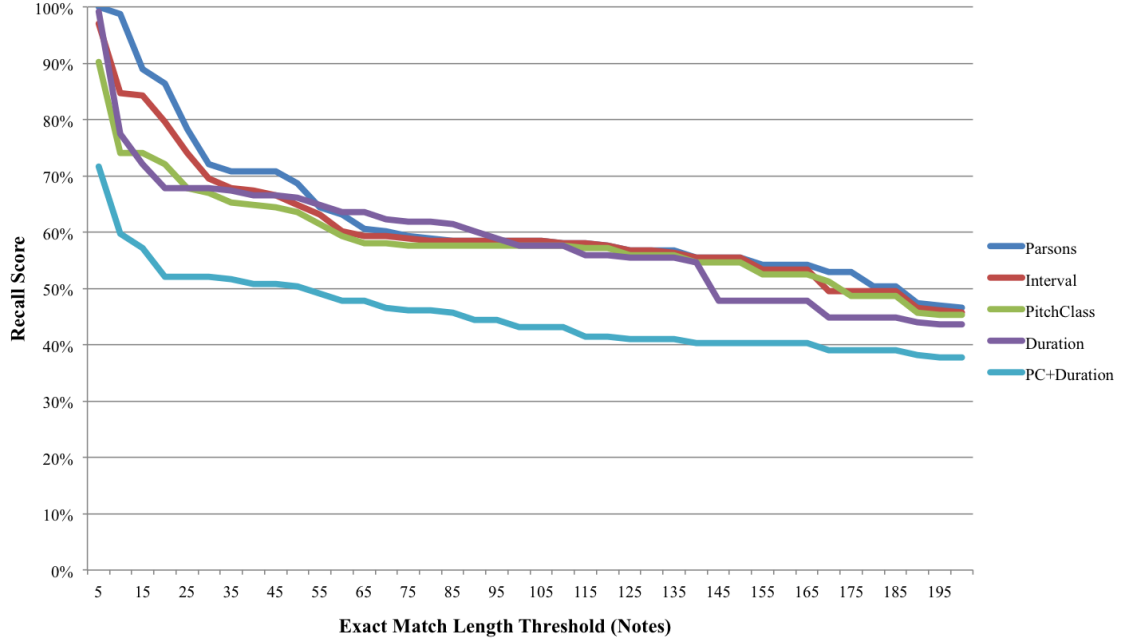


Figure 4.8: Recall scores of exact matching for various melodic representations in cover song identification on Lakh Covers dataset

The F_1 scores of each encoding for various exact match lengths are shown in Figure 4.9. Here again we see improved scores for PC+Duration at shorter lengths than the other encodings, although at greater lengths, this encoding’s performance worsens overall. The F_1 score is the balanced harmonic mean of the precision and recall scores, and consequently offers a conservative average of the precision and recall scores for the model at the specified threshold.

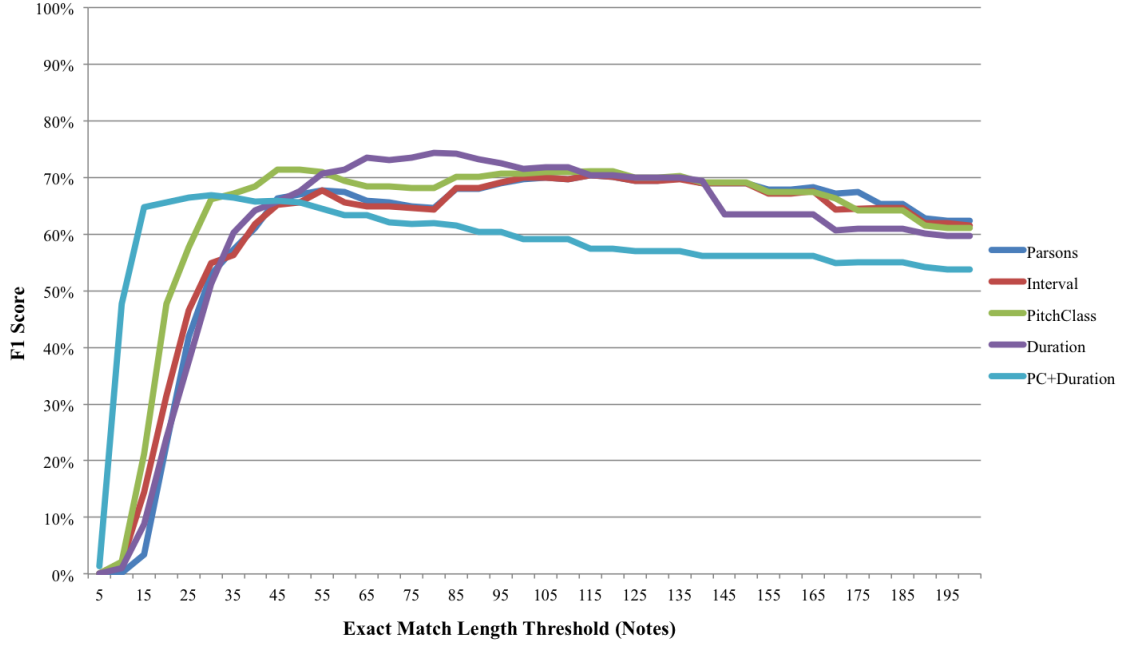


Figure 4.9: F_1 scores of exact matching for various melodic representations in cover song identification on Lakh Covers dataset

4.2.2 Inexact Matching

Global Alignment

Cover song identification using global alignment was evaluated using the edit ratio, described in Equation 3.5 of Section 3.5.1, as the threshold for the precision, recall, and F_1 scores. Edit ratio is plotted with descending values, as the greater the edit ratio, the greater the dissimilarity between two melodies in a given alignment or comparison. The same Lakh Covers dataset was applied with the same ground truth data to evaluate the effectiveness of global alignment in cover song identification using binary classification.

Precision scores for the five melodic encodings are shown in Figure 4.10. Similar to the trend with exact matching, all encodings achieve high precision scores, but the PC+Duration encoding does so at a considerably higher edit ratio threshold.

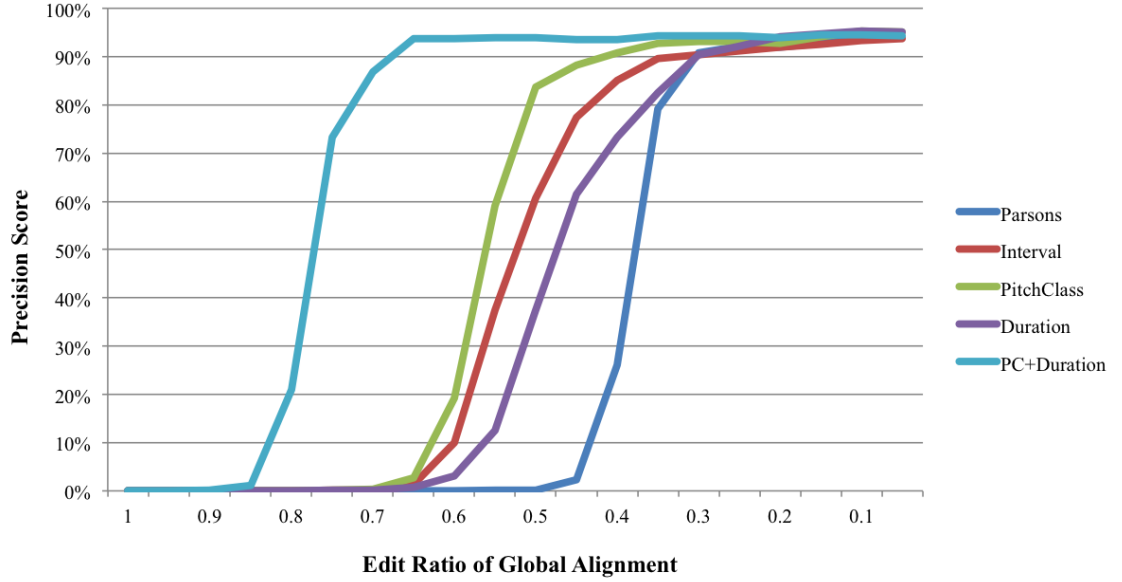


Figure 4.10: Precision scores for global alignment of various melodic representations in cover song identification on Lakh Covers dataset

The recall performance of each encoding is shown in Figure 4.11. Parsons and Interval encodings maintain near-perfect recall as the threshold decreases initially. These encodings represent the melody using inter-note relationships as opposed to the discrete note values. The Duration encoding also maintains this high performance initially. The PC+Duration encoding underperforms compared to the other encodings, in keeping with the trend observed with exact matching. Another interesting observation is that the encodings that are key-invariant (i.e. do not require transposition prior to exact or inexact matching) hold their initial recall scores, and

perform considerably better overall, than those encodings that require transposition.

This is likely due to some inefficacy inherent in the key-finding algorithm.

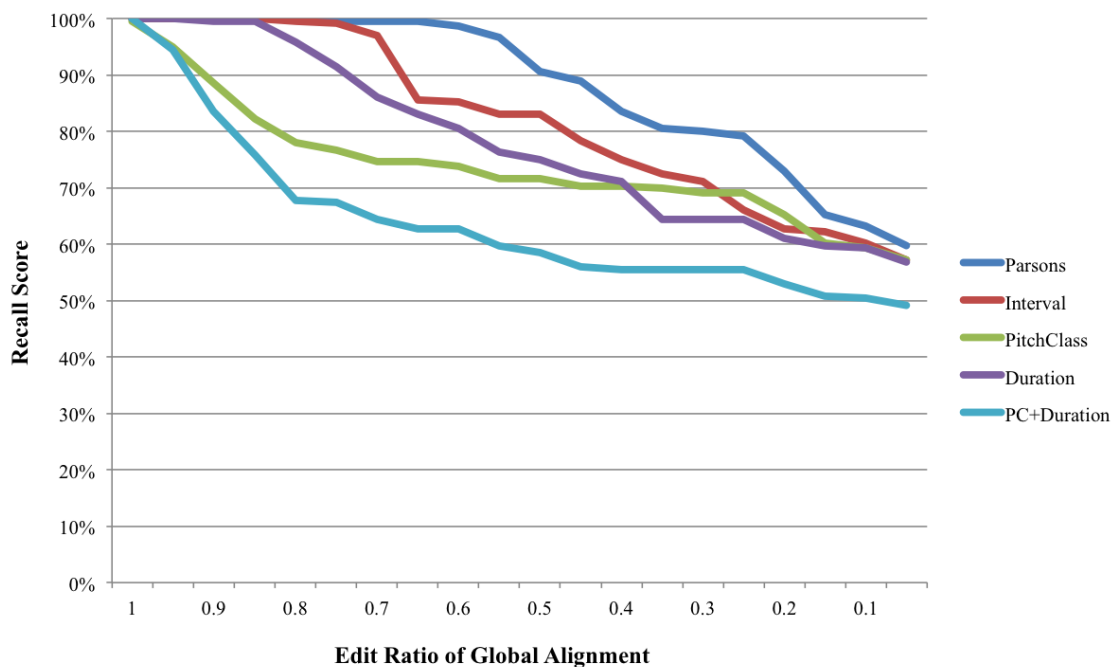


Figure 4.11: Recall scores for global alignment of various melodic representations in cover song identification on Lakh Covers dataset

The F_1 scores of the five encodings are illustrated in Figure 4.12. As seen in exact matching, we expect to see better performance earlier on from the encodings with greater alphabet sizes, a worsening in performance as the edit ratio decreases. Here we observe that Parsons achieves the highest overall score, outperforming the best F_1 score for exact matching. As in the exact matching F_1 results, PC+Duration achieves the best initial scores, but its performance worsens, in this case, as the threshold decreases

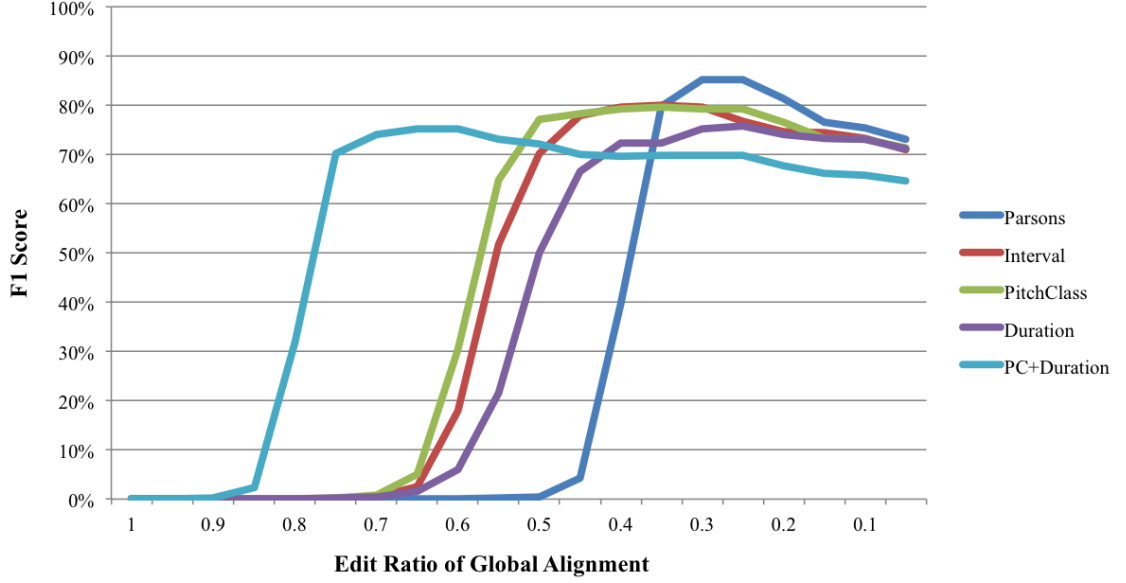


Figure 4.12: F_1 scores for global alignment of various melodic representations in cover song identification on Lakh Covers dataset

Local Alignment

Applying local alignment to cover song identification was performed by varying two distinct thresholds: the edit ratio of the alignment, described in Equation 3.5 of Section 3.5.1, and the minimum length of the local alignment. For the case of the minimum length of the local alignment, any alignments that did not meet the specified minimum length were ruled as being distinct melodies (i.e. not a cover or rendition). The impetus for varying the minimum alignment length as a threshold in the binary classifier is due to the observation that short lengths (e.g. less than ten notes) exhibit precision scores that often suffer greatly at any edit ratio.

Due to the evaluation of two separate thresholds for precision, recall and F_1 data,

there was considerably more information to visualize. Consequently, only the most salient figures will be discussed in this section, and all accompanying figures can be found in Appendix A. There is a notable difference in the alignment lengths of the different encodings, particularly so for Parsons. Table 4.1 shows the mean, median, and mode of the alignment lengths for all encodings in the Lakh Covers dataset. Consequently, different minimum alignment length thresholds needed to be applied for different encodings.

	Mean	Median	Mode
Parsons	226.6	229	231
Interval	20.9	13	6
PitchClass	20.0	14	8
Duration	7.4	5	3
PC+Duration	2.3	2	2

Table 4.1: Mean, median, and mode of local alignment lengths for the various encodings

Figure 4.13 shows the precision scores for various alignment lengths across all edit ratios. All minimum alignment lengths begin to perform at an edit ratio of about 0.35, although there is considerable diversion in the performance as the alignment length increases.

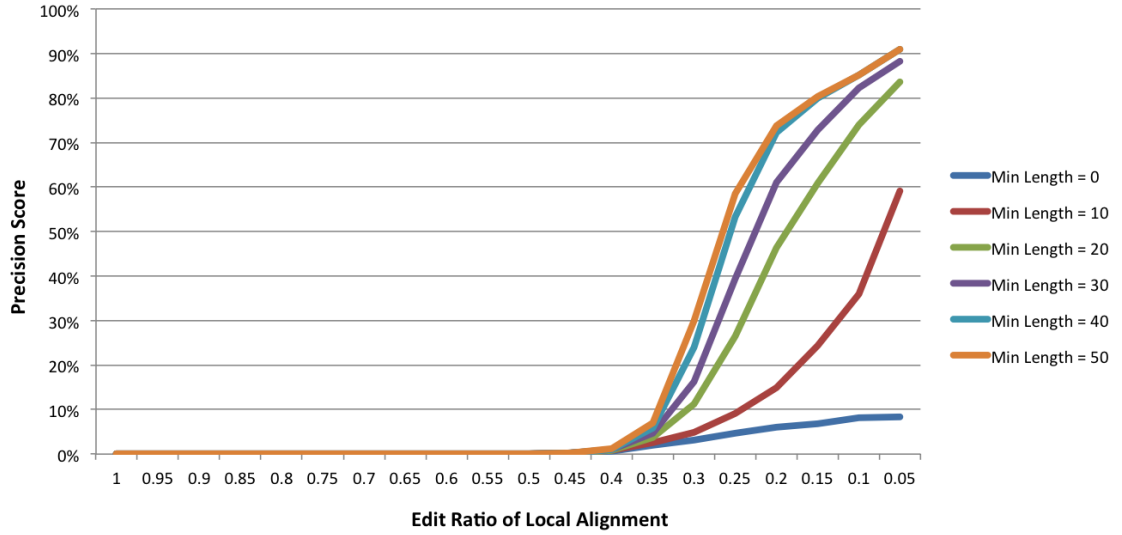


Figure 4.13: Precision scores of local alignment using Parsons encoding for various alignment lengths in cover song identification on Lakh Covers dataset

The precision scores for Interval and PitchClass encodings are very resemblant of each other, and are illustrated in Appendix A, Figures A.2 and A.3 respectively. A similar trend of improved performance with increasing alignment length was observed. The Duration encoding’s precision performance is also rather similar, and is shown in Figure A.4. All of these encodings begin to perform at an edit ratio between 0.45 to 0.4, and this range is also the point of divergence based on minimum alignment length.

For the PC+Duration encoding, we see improved performance in precision from the outset, which is resemblant of the trend in global alignment; however, in the global alignment precision scores, PC+Duration begins to perform at a considerably higher edit ratio threshold. Figure 4.14 shows the precision scores using local alignment of the PC+Duration encoding for various alignment lengths and edit ratios. The observed

behaviour of improved initial precision scores with greater minimum alignment length thresholds suggests that the classifier is casting a wide net, and will likely suffer in recall performance as the alignment length threshold increases.

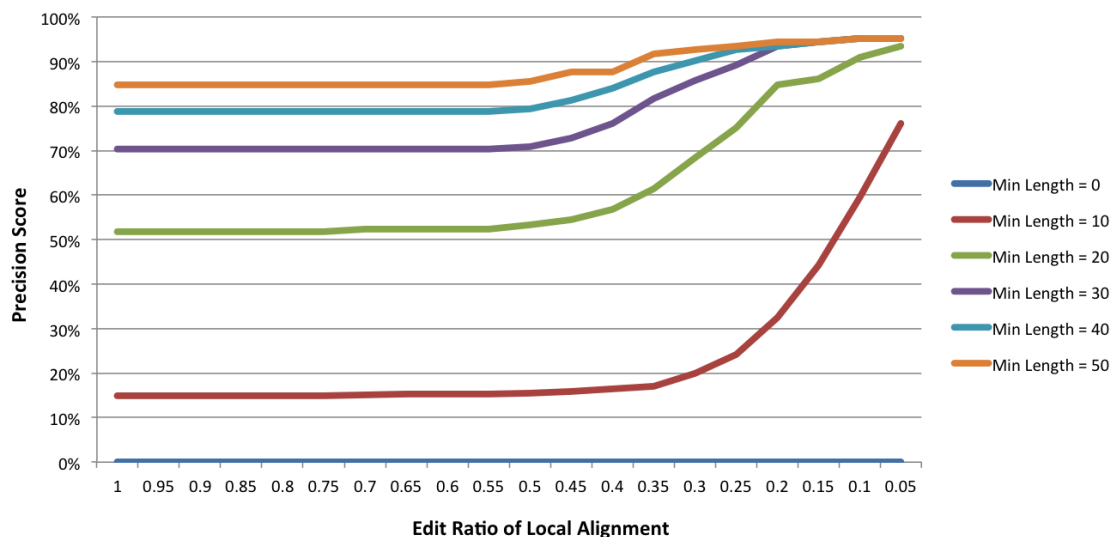


Figure 4.14: Precision scores of local alignment using PC+Duration encoding for various alignment lengths in cover song identification on Lakh Covers dataset

As noted earlier in this section, the match lengths differ greatly for each encoding, particularly for Parsons. In light of this, different minimum length thresholds were applied to different encodings to illuminate this phenomenon and provide a more detailed understanding of the variations in predictive performance as the thresholds are modified. Figure 4.15 shows the recall scores for Parsons with different alignment lengths and edit ratios. Shorter alignment lengths perform better overall; however, at smaller edit ratios, all match lengths perform worse and the difference in performance based on match length is diminished. It should be noted that while a minimum alignment length of zero is not plotted, the series was identical to the minimum

alignment length of 50, and as such, this series was plotted for convenience.

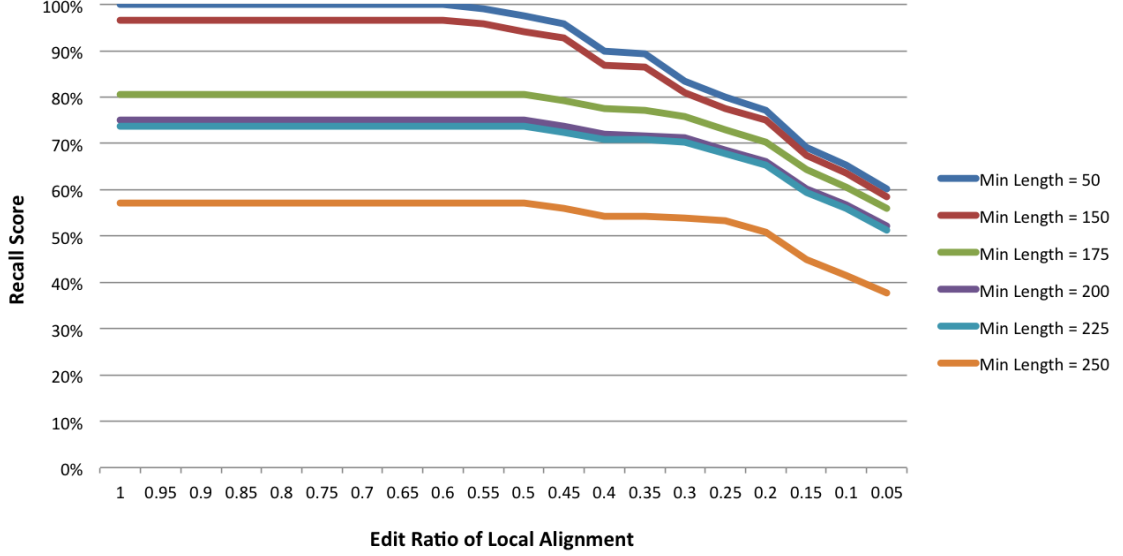


Figure 4.15: Recall scores of local alignment using Parsons encoding for various alignment lengths in cover song identification on Lakh Covers dataset

The difference in performance as a result of varying the alignment length in Interval encoding is very minimal, compared to the other encodings. This is shown in Figure A.7 in Appendix A. PitchClass and Duration encodings follow a similar trend as those displayed in Parsons and Interval encodings, where edit ratios less than 0.5, as well as increases in alignment length, begin to diminish recall scores. The recall scores of these encodings are shown in Figures A.8 and A.9.

Notably, PC+Duration shows almost no decline in recall scores as a consequence of smaller edit ratios. However, increasing the minimum alignment length most negatively affects the PC+Duration encoding. There is perhaps very little improvement to be gained from an encoding with such high specificity. This behaviour can be

seen in Figure 4.16, which illustrates the recall scores of local alignment with the PC+Duration encoding.

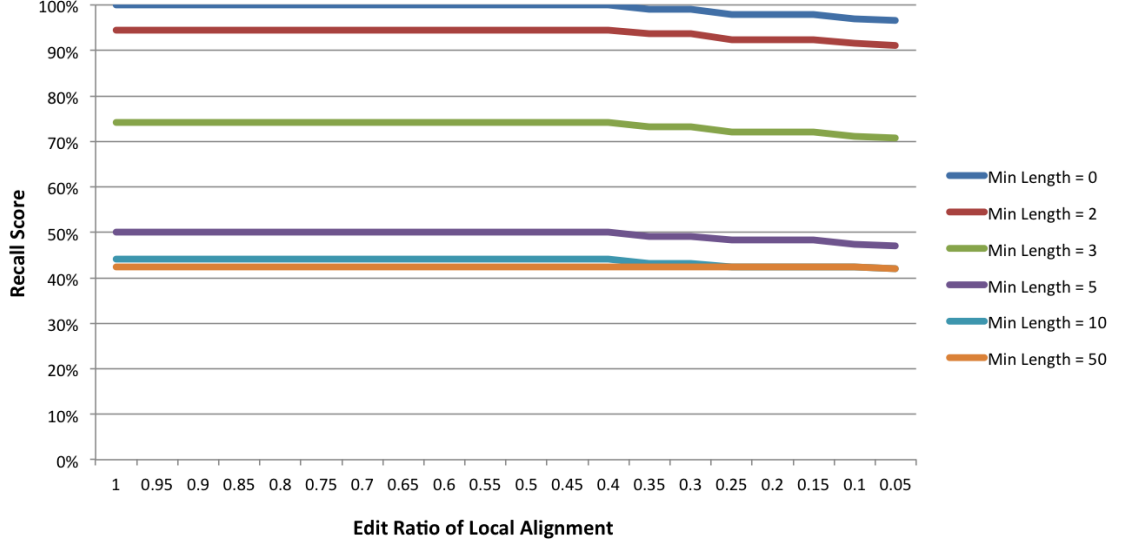


Figure 4.16: Recall scores of local alignment using PC+Duration encoding for various alignment lengths in cover song identification on Lakh Covers dataset

With precision, we observe that increases in minimum alignment length improve the classification performance, whereas in recall, the performance diminishes as the minimum alignment length increases. The extent to which this is true depends in part on the encoding. We will examine the overall effect of this further by evaluating the F_1 scores for all encodings.

F_1 scores were evaluated for the various edit ratio and minimum alignment length thresholds for all melodic encodings. In keeping with precision and recall, the F_1 scores of Interval, PitchClass, and Duration encodings are nominally different, and are shown in Figures A.12, A.13, and A.14 respectively. Parsons encoding marginally achieves the best overall F_1 score, but perhaps more importantly, it achieves suitably

better F_1 performance than the other encodings for any given minimum alignment length threshold. The F_1 scores for Parsons can be seen in Figure 4.17.

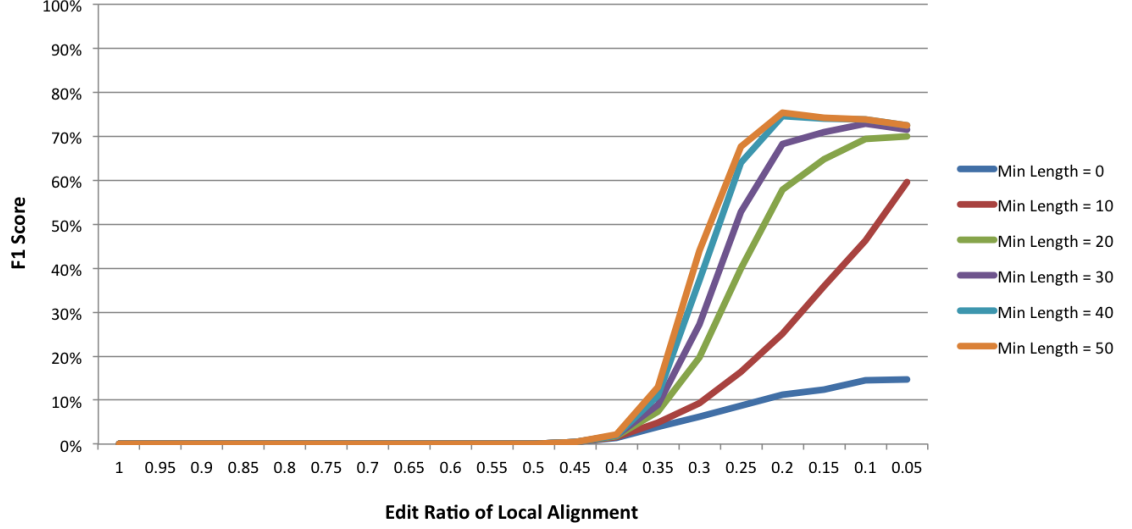


Figure 4.17: F_1 scores of local alignment using Parsons encoding for various alignment lengths in cover song identification on Lakh Covers dataset

The PC+Duration encoding performs poorly for all length thresholds save for minimum alignment length = 10. Figure 4.18 suggests minimum alignment length thresholds between 0 and 20 may also perform suitably well compared to the other encodings. However, overall PC+Duration achieves the poorest F_1 performance at greater alignment length thresholds.

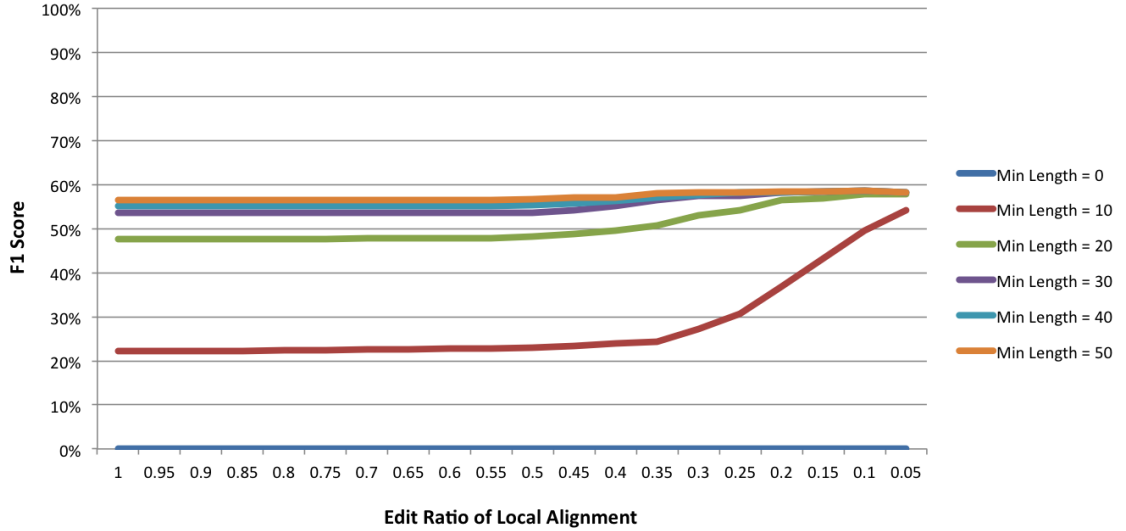


Figure 4.18: F_1 scores of local alignment using PC+Duration encoding for various alignment lengths in cover song identification on Lakh Covers dataset

4.3 Cover Song Identification: Jazz Standard

A dataset of seven renditions of the George Gershwin jazz standard *Summertime* was compiled and the melodies were extracted. This dataset was used to evaluate the effectiveness of exact and inexact matching in cover song identification across the various melodic representations.

The covers group consisted of 21 melodic comparisons of 7 unique renditions of *Summertime*, and the control group consisted of 21 melodic comparisons of 7 unique popular music songs chosen randomly from the Lakh Unique Songs dataset. The acquisition of these datasets is described in Section 3.3.

4.3.1 Exact and Inexact Matching

For exact matching, the LCS was solved using Suffix Trees (ST), and the data used to perform the Wilcoxon signed-rank tests were the exact match lengths of the covers group and the control group. The Wilcoxon test was configured to discard zero-differences in the ranking process, and consequently some results have insufficient sample sizes for normal approximation. These entries are colored in orange within the table. For inexact matching, global alignments (GA) and local alignments (LA) were performed and the edit ratios of these alignments were used to perform the tests. Table 4.2 describes the p -values of the Wilcoxon tests for exact and inexact matching across all melodic encodings.

	ST - Exact	GA - Inexact	LA - Inexact
Parsons	0.0008	0.0582	0.0001
Interval	0.0284	0.3219	0.6378
PitchClass	0.0321	0.0106	0.7943
Duration	0.5863	0.7413	0.3424
PC+Duration	0.0675	0.4342	inf.

Table 4.2: p -values of Wilcoxon signed-rank tests for various melodic representations in jazz cover song identification

We can see that Parsons is the only encoding to perform near or below the 5% alpha risk (i.e. a confidence level $> 95\%$) for all three matching approaches. However, the global alignment test results can be considered meaningless for Parsons, as the median alignment edit ratio for the control group is less than that of the covers group. The Interval encoding with exact matching achieves a p -value below 0.05 significance, as well as the PitchClass encoding for exact matching and inexact matching using

global alignment.

4.3.2 Local Alignment with Minimum Length Threshold

In this Section, we explore the addition of varying minimum alignment length thresholds to observe the effect on the predictive accuracy of inexact matching using local alignment in jazz cover song identification. As observed in Section 4.2.2, the introduction of a minimum alignment length when using local alignment can improve the predictive accuracy of cover song identification in pop music for some encodings at certain edit ratio thresholds. Table 4.3 shows the p -values of the Wilcoxon tests comparing 21 jazz covers and 21 unique melodic comparisons for various minimum local alignment length thresholds. Specifically, if a local alignment achieved a length less than the minimum length threshold, the edit ratio was assigned a value of one (i.e. most dissimilar). From Table 4.1, we can see that different encodings have different mean alignment lengths, and consequently will respond to these thresholds differently.

	Minimum Length Threshold (Notes)							
	0	2	5	10	20	50	100	200
Parsons	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.9584	0.1221
Interval	0.6378	0.6378	0.7437	0.0170	0.0357	0.1088	inf.	inf.
PitchClass	0.7943	0.7943	0.7943	0.0117	0.0005	0.0050	inf.	inf.
Duration	0.3424	0.1846	0.5700	0.8335	1.0000	inf.	inf.	inf.
PC+Duration	inf.	0.1573	0.3173	inf.	inf.	inf.	inf.	inf.

Table 4.3: p -values of Wilcoxon signed-rank tests for inexact matching using local alignment and applying different minimum alignment length thresholds

Overall, Parsons achieves an alpha risk sufficiently below 0.05 for length thresholds

below 100; however, for minimum length thresholds of 20 and 100, there are insufficient samples after discarding zero-differences. Additionally, for minimum length thresholds of 100 and 200, the Parsons control group has greater mean and median lengths than the experimental (i.e. covers) group. This suggests these values are erroneous. The Interval and PitchClass encodings achieve p -values slightly below 0.05 at minimum alignment lengths of 10 and 20. These findings are consistent with the mean and median lengths shown in Table 4.1 for the Interval and PitchClass encodings, suggesting that there is a “sweet spot” for each encoding based on the amount of information it encodes.

4.4 Jazz Cover Song Identification with Varying Occurrence

In order to better establish the predictive accuracy of exact and inexact matching in jazz cover song identification, we sought to evaluate the performance using binary classification while varying the occurrence of covers within the dataset. For this particular application the Summertime Jazz Standard dataset was configured into three separate datasets. The first contained 14 melodies: 7 unique melodies, and 7 renditions of *Summertime*, or a 50% occurrence of jazz covers. The other two datasets had a jazz cover occurrence of 10% and 1% respectively. In order to minimize the amount of data to visualize, only Parsons, Interval, and Duration encodings were

evaluated and only the F_1 scores were summarized. All these results are summarized in Appendix B, Figures B.1 to B.9. Some of the results are discussed below; however, in all cases, the F_1 scores for all encodings using both exact and inexact matching with a jazz cover occurrence of 1% were essentially zero and lacking any predictive value.

Figure 4.19 shows the F_1 scores of Parsons encoding using exact matching for various occurrences of jazz cover songs. While the F_1 scores for the 50% occurrence data were promising, as the occurrence decreases, the predictive power diminishes greatly. This trend is true of all encodings investigated. Interestingly, the Duration encoding performs better than Parsons or Interval encodings at 50% occurrence of covers. The F_1 scores of the Duration encoding for jazz cover song identification can be seen in Figure B.3.

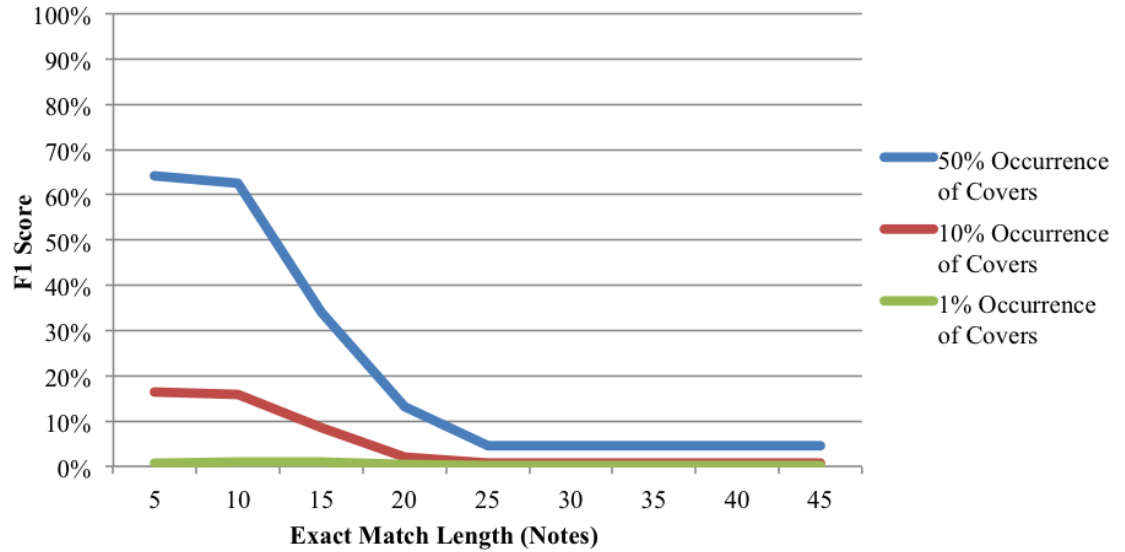


Figure 4.19: F_1 scores of exact matching using Parsons encoding with different occurrences of covers in the jazz Standard dataset

Figures 4.20 and 4.21 depict the inexact matching F_1 scores using Parsons encoding for global alignment and local alignment respectively. Both perform modestly better at lower edit ratio thresholds than the exact matching approach for Parsons at 50% and 10% occurrence of jazz covers. At 1% occurrence, essentially all approaches achieve negligible F_1 scores.

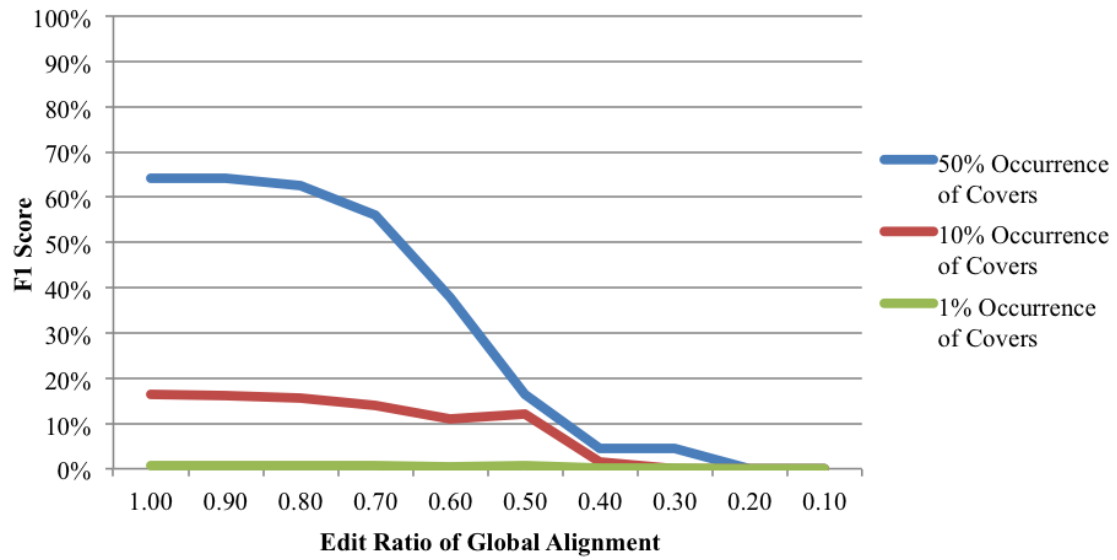


Figure 4.20: F_1 scores of global alignment using Parsons encoding with different occurrences of covers in the jazz Standard dataset

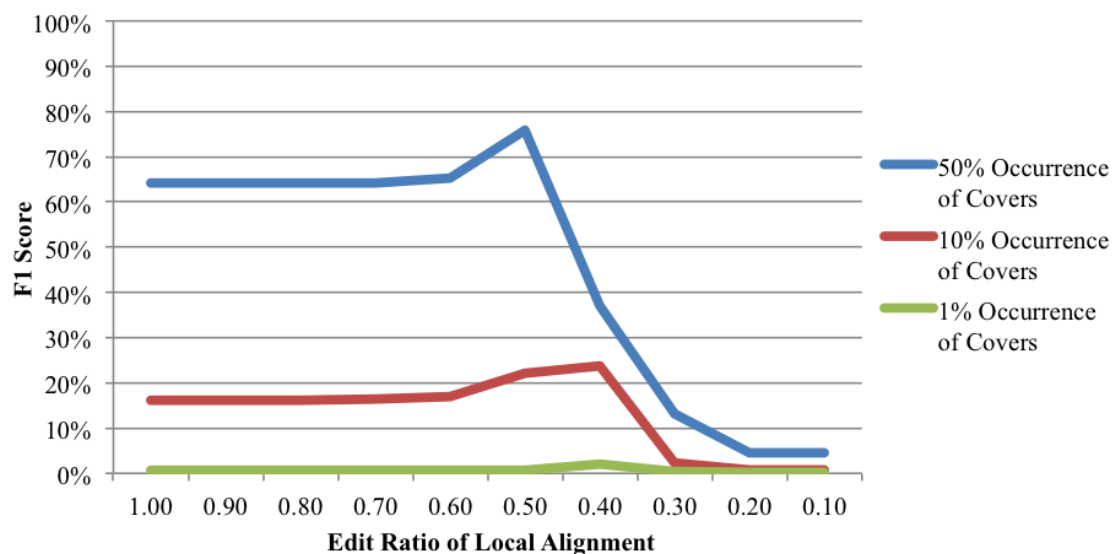


Figure 4.21: F_1 scores of local alignment using Parsons encoding with different occurrences of covers in the jazz Standard dataset

Overall we can infer there is too much variation in jazz melodies to provide meaningful differentiation between covers and unique songs using these fairly naïve classifiers. It is difficult to trust the data within the smaller datasets (i.e. greater occurrence of covers) in view of the overwhelming consensus of poor predictive performance in larger datasets.

Chapter 5

Conclusions

This chapter will discuss the findings of the previous chapter, outline how the thesis statement has been demonstrated, and explore possible explanations for the observations in this work. This thesis has demonstrated that substring matching techniques, particularly suffix trees and dynamic programming, are viable methods for evaluating symbolic (i.e. transcriptional) similarity of melodies in pop music; whereas they are insufficient on their own for genres such as Jazz, where the melodic and rhythmic variations are much greater.

5.1 Unique Songs

The application of the Lakh Unique Songs dataset in this work is chiefly to investigate the behaviour of the five encodings with respect to the data available. In this regard, there is no investigation or discussion of the predictive power or accuracy

of specific configurations in this particular set of tests. Instead, we simply establish some expectations for performance of the algorithms in later tests and contexts.

The distributions and behaviour of Interval and PitchClass encodings were quite similar for both exact and inexact matching. Duration presented with very similar results for exact matching and global alignments; although, in local alignments the Duration encoding proved to embody more specificity than Interval or PitchClass encodings. For exact matching, Parsons exhibits the broadest distribution with the lowest peak and it is the encoding whose distribution peaks at the longest match length. For global alignment, the Parsons distribution peaks with the lowest edit ratio, implying this representation requires the fewest edits to globally align the melodies in this dataset. However, all distributions peaked at edit ratio values > 0.5 for global alignment, which is generally considered insignificant for a pairwise sequence alignment [61].

For exact and inexact matching, PC+Duration behaves in essentially opposite fashion to Parsons insofar as its distributions are tallest and narrowest, it peaks with the shortest match length of all encodings for exact matching, and it maintains the highest edit ratio of all encodings for global alignment. For local alignment, Parsons is the only distribution to somewhat resemble a normal distribution, and may indicate the generality of this representation as compared to the others.

We observe a trend in the Smith-Waterman local alignment algorithm that the greater the specificity (i.e. larger the alphabet) the shorter the length of match and

the fewer edits for a given match. This creates a focus on short regions of high similarity. Often these regions were very low-complexity, for example five repeated notes, and bore little significance in terms of the alignment match. Instead, longer regions of moderate similarity are often more significant than short regions of high identity [62]. The extent to which this tendency holds true is subject to the nature of the data and the parameters of the local alignment implementation.

5.2 Cover Song Identification: Pop

The application of binary classification to evaluate cover song identification within the Lakh Covers dataset is intended to evaluate the performance of exact and inexact matching in the context of information retrieval. Specifically we plan to interrogate the predictive capacity of the five encodings at identifying covers of pop song melodies. As well, we plan to identify thresholds of key parameters that can be manipulated to improve outcomes based on domain-specific knowledge. Ground truth data was established by identifying all renditions of each song in the dataset either by the same artist or a different artist.

The PC+Duration encoding exhibits the best precision scores at the shortest lengths for exact matching; however, it performs the worst at all match lengths in recall and its F_1 scores suffer as a consequence. Overall, it is still the best encoding for exact matching queries with lengths under 25 notes in terms of F_1 performance. Duration achieves the peak F_1 score of all encodings, but this is done at match lengths

of approximately 80 notes.

For global alignment, PC+Duration exhibits far better precision scores at higher edit ratio thresholds than the other encodings; however, it again produces the worst recall scores at any edit ratio threshold. The precision scores for Duration rise at lower edit ratio thresholds, along with Parsons, suggesting the predictive power of this encoding is greater in shorter matches or alignments, and suffers at greater alignment lengths. Parsons achieves the best overall F_1 scores, and it reaches its peak F_1 performance at an edit ratio threshold lower than any other encoding and sufficiently lower than 0.5.

Local alignments show a dramatic difference in alignment lengths between Parsons and the other encodings. This perhaps suggests the effect of a small alphabet and uniform penalties on the Smith-Waterman algorithm is very significant. Conversely, with PC+Duration the size of the alphabet, and naturally the greater specificity, drives shorter alignment lengths with Smith-Waterman; so much in fact, that it hinders the F_1 scores of the PC+Duration encoding for nearly all thresholds of edit ratio and minimum alignment length. The PC+Duration encoding in local alignment is the most negatively affected by small increases in minimum alignment length thresholds; however, it is almost entirely unaffected by edit ratio thresholding, due to the nature of emphasizing short alignments with high similarity.

The chief observation for local alignments in pop music is that providing longer melody segments allows encodings with less information to classify with greater ac-

curacy than shorter melody segments encoded with more information. The extent to which this observation is true depends on the encodings and the melodies.

5.3 Cover Song Identification: Jazz

The Wilcoxon ranked-sum tests for exact and inexact matching showed some promising results, however there were issues with some of the results that bear discussion. Namely, the global alignment test value for Parsons was significant yet the median edit ratio was smaller in the control group than in the covers group. Encodings with smaller alphabet sizes performed better overall as compared to those encodings with larger alphabets. Exact matching generally outperformed inexact matching scores for global alignment and local alignment without a minimum length threshold. Yet, when applying minimum length thresholds on the order of approximately 10 to 20 notes, the local alignment Wilcoxon test scores improved significantly for Interval and PitchClass encodings.

In order to investigate how the Wilcoxon test results may reflect an encoding’s capacity to identify cover songs in jazz, we performed Precision, Recall, and F_1 scores for three datasets of different sizes that contained the *Summertime* covers. The intention was to observe how these encodings perform with a diminishing presence of covers in a dataset of otherwise unique songs.

The chief observation was that at 1% occurrence, the predictive power of any encoding investigated, using either exact or inexact matching, was negligible in iden-

tifying jazz covers from other unique melodies. We observe that Duration produces the best F_1 scores for exact matching at 50% occurrence of covers. At 10% occurrence, Parsons performs well at short match lengths; however, Duration achieves greater F_1 scores at longer match lengths. For global and local alignments, Parsons achieves the best F_1 scores for 50% and 10% occurrences of covers in the datasets. Ultimately, the effect of growing the dataset with unique songs is overwhelmingly the largest determinant in classification performance for all of the measures tested. It is difficult to put a lot of weight behind the observations we see at 50% or even 10% occurrence in light of this phenomenon.

Some measures could be taken to potentially improve the outcomes of these classification tasks. In particular, we could explore techniques to remove embellishments, or employ a substitution matrix with reward schemes that are specific to the domain (i.e. the genre or corpus). Overall, the results suggest the breadth of rhythmic and melodic variation present in jazz melodic improvisation is so diverse that these fairly naïve algorithms get stuck on the embellishments and cannot differentiate the salient notes that make up the core melody or “hook” of the standard. For this reason, higher level analysis or processing would likely need to be employed to improve the classification performance with this and other more improvised idioms.

5.4 Contributions

This thesis has demonstrated that exact and inexact matching techniques can be applied to symbolic melodic similarity successfully. Some further contributions of this work are discussed herein. A large dataset of monophonic melodies was generated from MIDI transcriptions for the analysis of this work. Ukkonen’s suffix tree was implemented and has been successfully applied to solve exact matching of melodic sequences in a variety of transcriptional forms. The results of exact and inexact matching approaches were analyzed in three distinct contexts: comparing unique melodies, pop cover song identification, and jazz cover song identification.

5.5 Future Work

5.5.1 Binary Classification

There are a number of potential improvements that could be applied to the task of binary classification to attempt to improve the classification performance of the encodings presented in this work and other symbolic representations of melodies. As well, there are additional tests that should be considered to produce further insights into the behaviour of these classifiers. Rank-based measures of classification such as Average Dynamic Recall (ADR) or Average Precision (AP) should be considered in place of set-based classification measures. Dataset(s) could be produced containing covers ranked by their similarity to the original. The ground truth of this data would

be subject to some strong assumptions but nevertheless this kind of testing could demonstrate important behaviour(s) of the algorithms employed.

Embellishments could be categorized and evaluated for their effect on exact and inexact matching algorithms. To this end, artificial data could be generated to simulate different kinds of melodic changes. Moreover, various techniques to eliminate, filter out, or minimize the effect of embellishing tones on the core melody should be explored in the context of more improvised forms of music. Some specific approaches may include: dropping shorter-duration notes, merging shorter-duration notes to their adjacent longer-duration note, quantizing longer-duration notes more coarsely to minimize the alphabet size, reducing the rendition to a contour, applying word-based approaches to further reduce the complexity of the symbolic representation, or correlating the melody to underlying harmonic structures.

With these and other efforts, it would be valuable to explore the effects of various key-finding algorithms on outcomes for classification. Many of these algorithms are very robust under certain assumptions or within certain genres; however, all of them have weak points that should be explored.

In place of MIDI transcriptions as the source data for these tests, it would be valuable to employ automatic melody extraction algorithms, such as MELODIA, on audio recordings, to obtain a symbolic representation that can be used in the current framework for testing.

5.5.2 Exact Matching

Suffix Trees and other exact matching algorithmic approaches to substring searching are not limited to solving the LCS problem. In this regard, there are a number of applications of Suffix Trees that may be used to identify similarities between two melodies. For example, the number of common substrings greater than a specified length could be counted for a given comparison of two melodies. In this work, as in many applications of Suffix Trees, many of the LCS solutions produced low-complexity substrings. In lieu of this, filtering techniques should be considered to reduce the effect of long sequences of low-complexity on the performance of the system.

5.5.3 Inexact Matching

For Inexact matching, the incorporation of a substitution matrix based on different scoring schemes (e.g. consonance, symbolic frequency, psychological perception, note duration, etc.) may improve the performance of these algorithms in identifying similar melodies. In bioinformatics, substitution matrices such as PAM and BLOSUM have been applied to amino acid or DNA sequence alignments, where domain specific knowledge was used to inform the costs of specific substitutions [63]. It is important to note that the basis for creating a substitution matrix will manifest itself in the output of the alignment scoring, and consequently substitution matrices can be created to fulfill specific needs or assumptions.

With respect to local alignment, likely the most important effort would be to alter

or fine tune the Smith-Waterman algorithm to focus on longer regions of moderate similarity and minimize the tendency to produce short alignments with few or no edits. The optimization of such an approach would hinge on the type of encoding being used and the expected alignment lengths of unique and similar songs.

Appendix A

Cover Song Identification of Pop Music Using Local Alignment

Precision

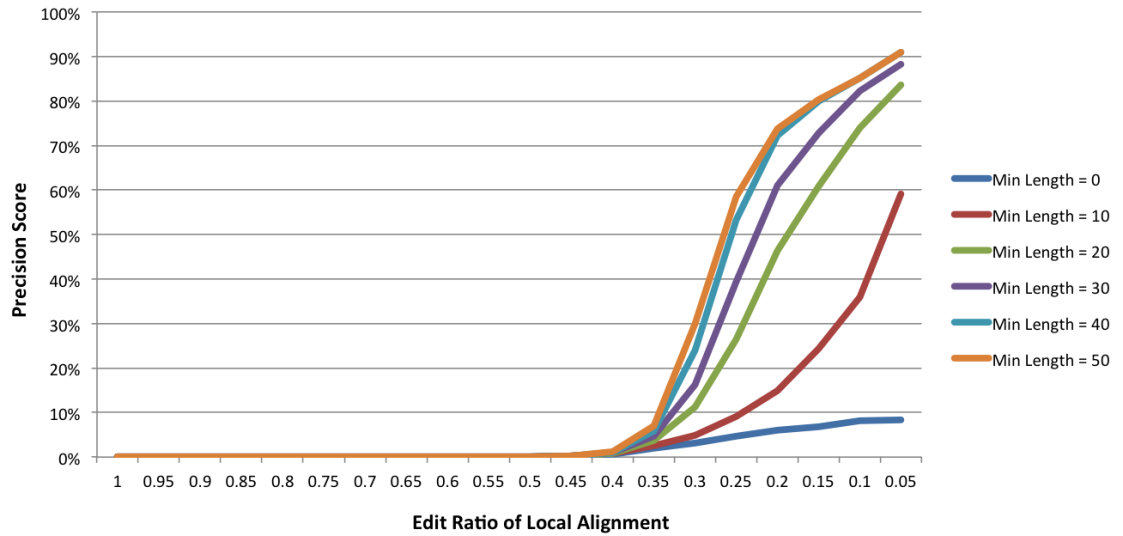


Figure A.1: Precision scores of local alignment using Parsons encoding for various alignment lengths in cover song identification on Lakh Covers dataset

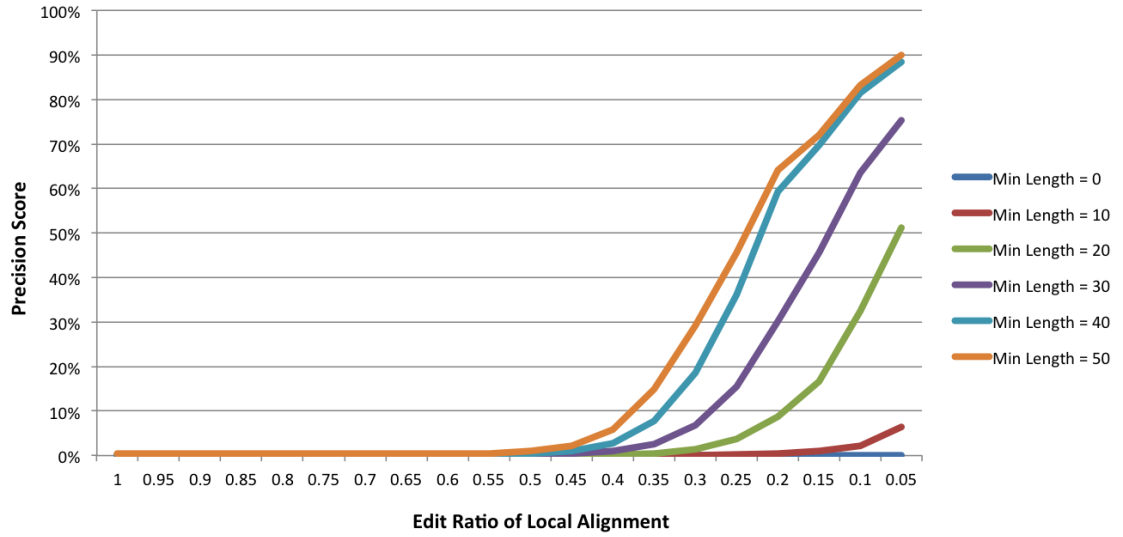


Figure A.2: Precision scores of local alignment using Interval encoding for various alignment lengths in cover song identification on Lakh Covers dataset

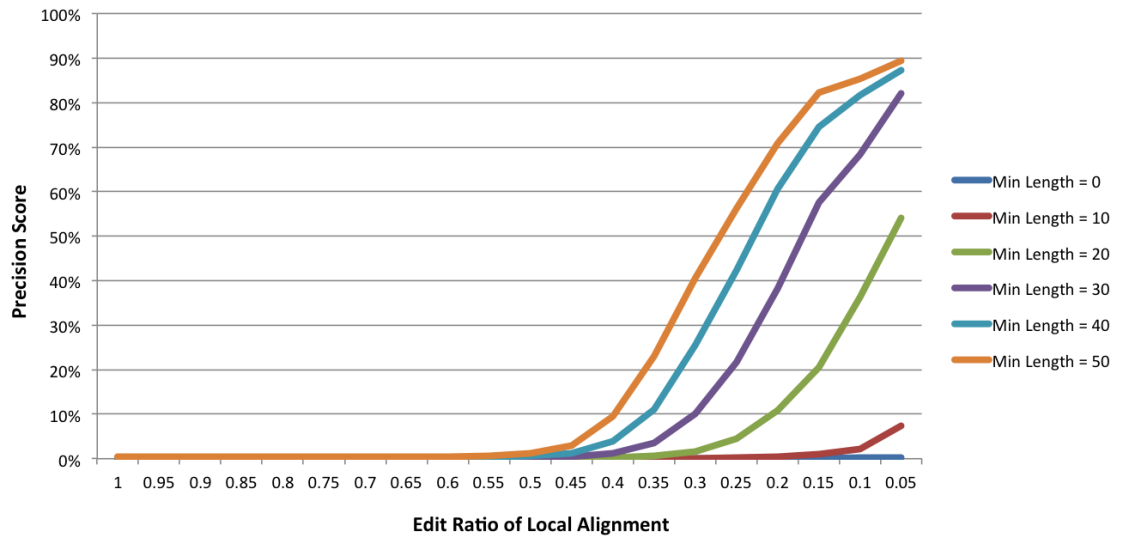


Figure A.3: Precision scores of local alignment using PitchClass encoding for various alignment lengths in cover song identification on Lakh Covers dataset

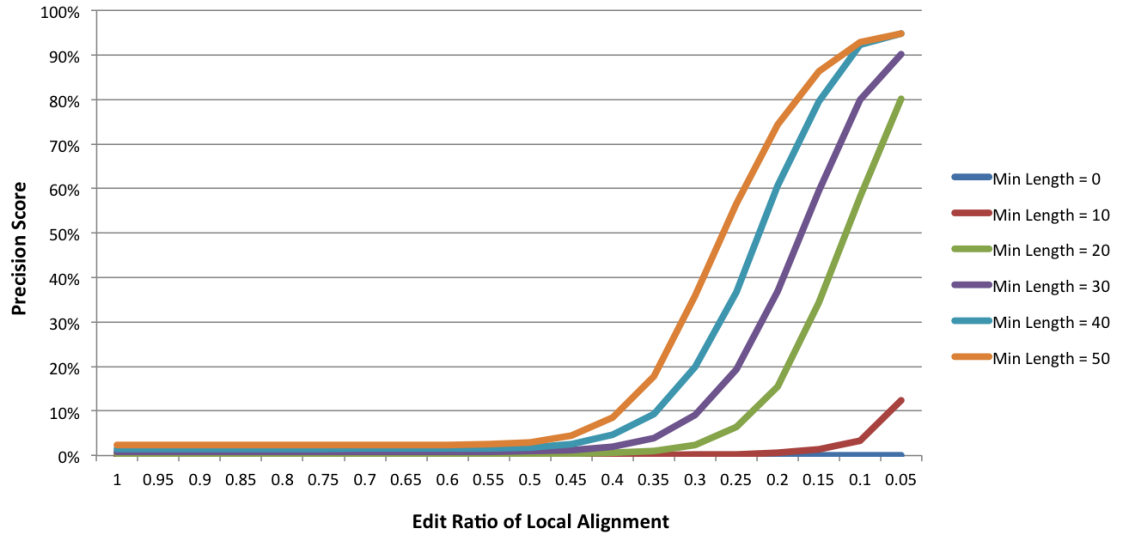


Figure A.4: Precision scores of local alignment using Duration encoding for various alignment lengths in cover song identification on Lakh Covers dataset

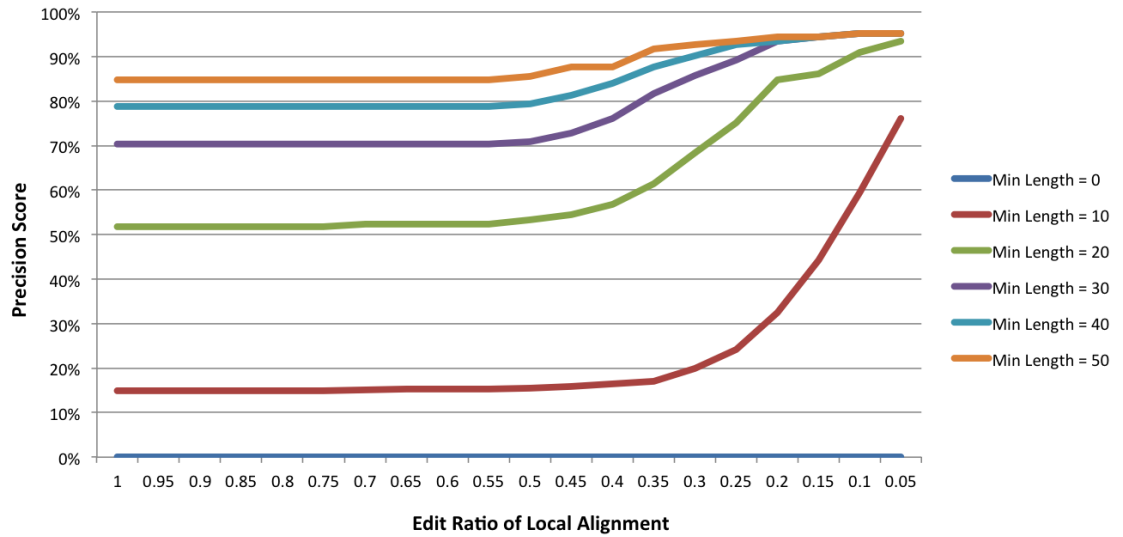


Figure A.5: Precision scores of local alignment using PC+Duration encoding for various alignment lengths in cover song identification on Lakh Covers dataset

Recall

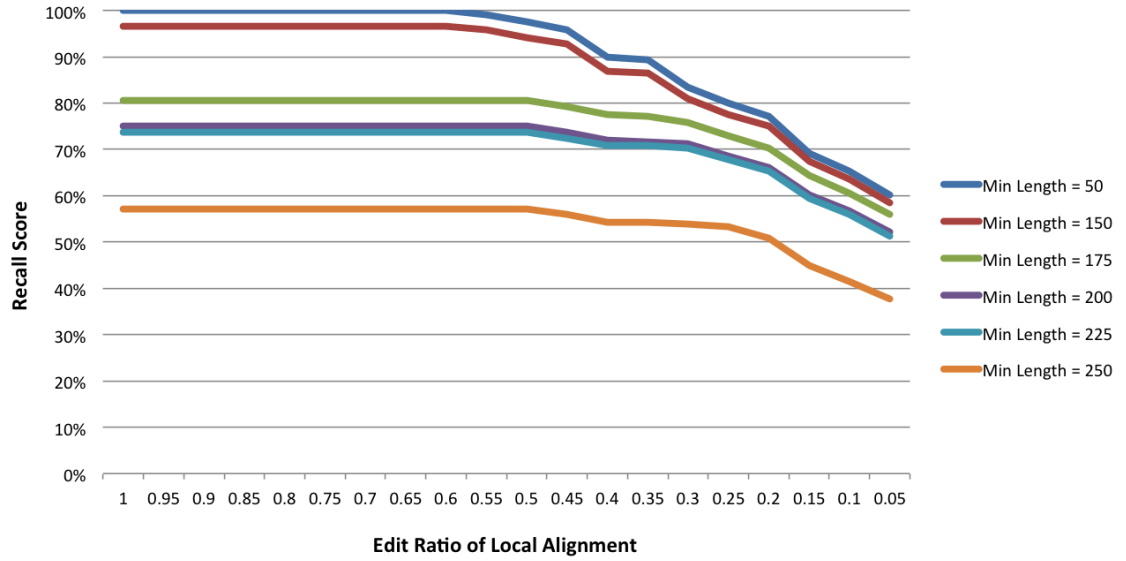


Figure A.6: Recall scores of local alignment using Parsons encoding for various alignment lengths in cover song identification on Lakh Covers dataset

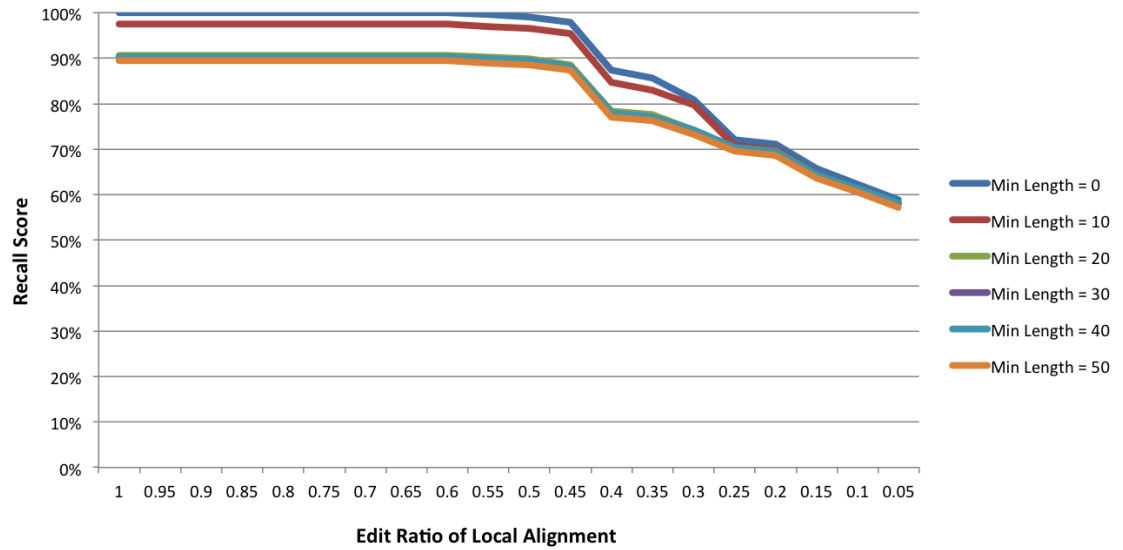


Figure A.7: Recall scores of local alignment using Interval encoding for various alignment lengths in cover song identification on Lakh Covers dataset

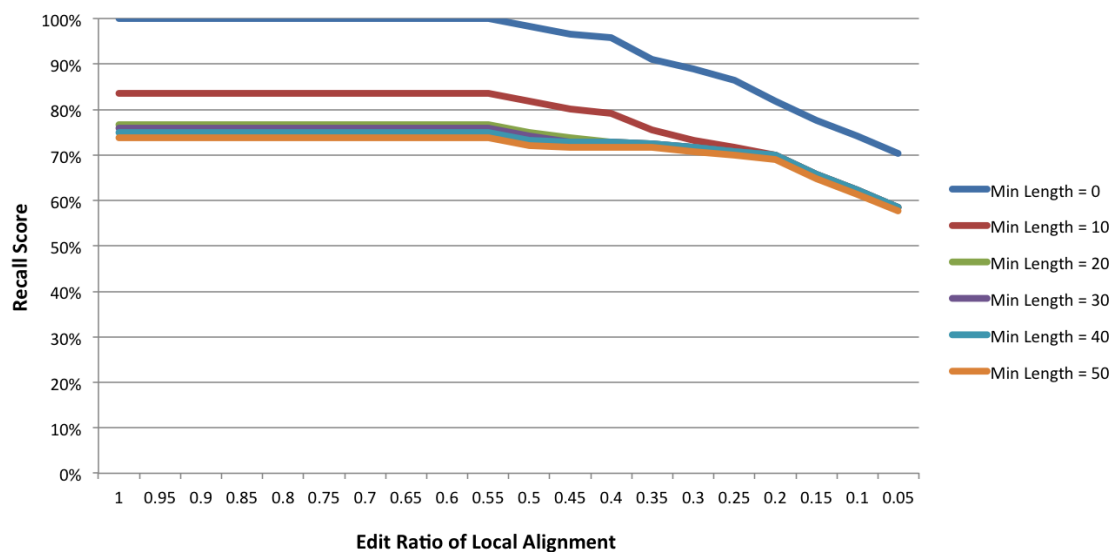


Figure A.8: Recall scores of local alignment using PitchClass encoding for various alignment lengths in cover song identification on Lakh Covers dataset

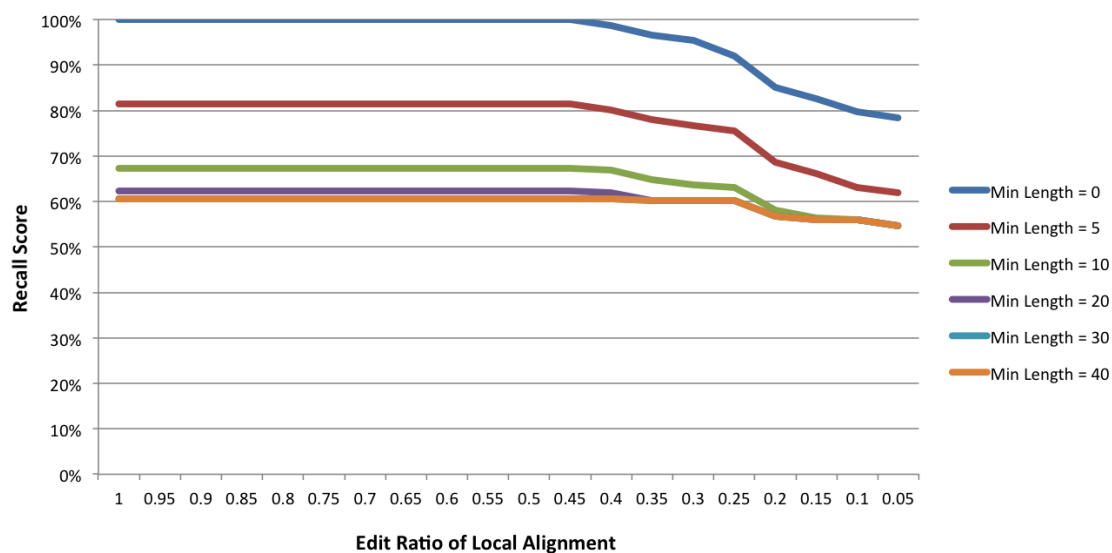


Figure A.9: Recall scores of local alignment using Duration encoding for various alignment lengths in cover song identification on Lakh Covers dataset

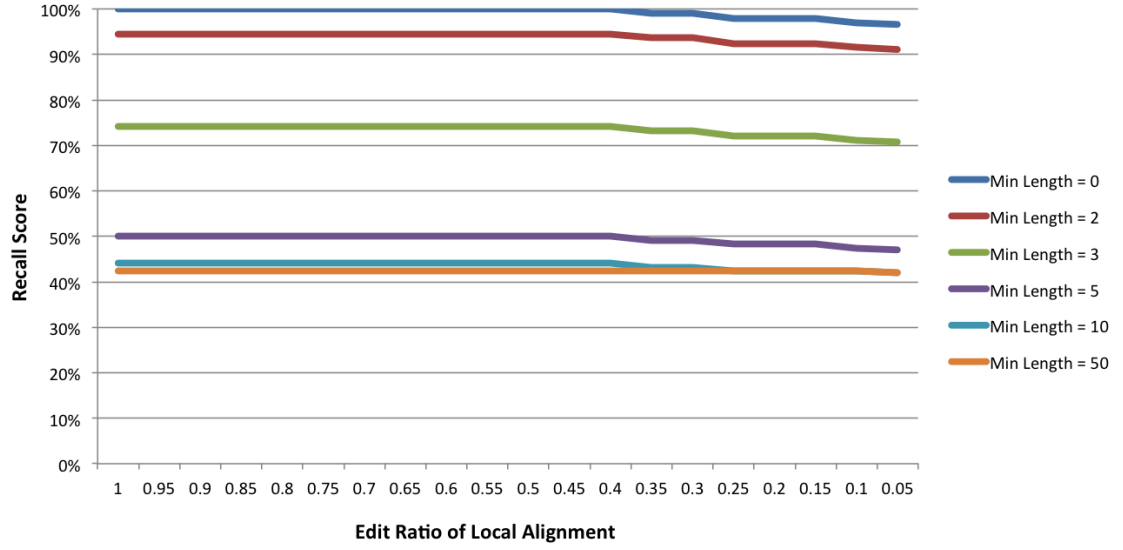


Figure A.10: Recall scores of local alignment using PC+Duration encoding for various alignment lengths in cover song identification on Lakh Covers dataset

F_1

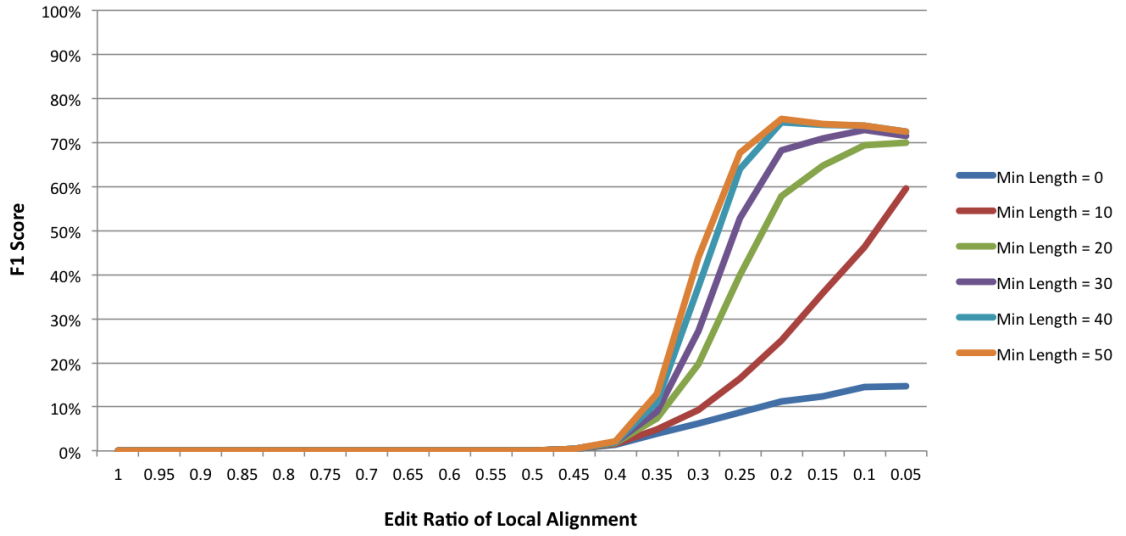


Figure A.11: F_1 scores of local alignment using Parsons encoding for various alignment lengths in cover song identification on Lakh Covers dataset

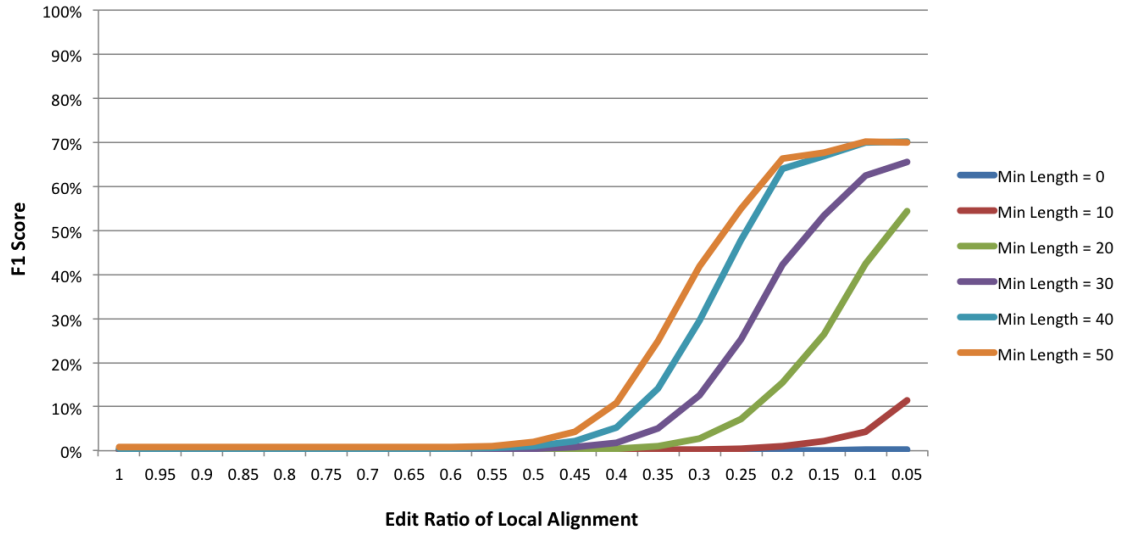


Figure A.12: F_1 scores of local alignment using Interval encoding for various alignment lengths in cover song identification on Lakh Covers dataset

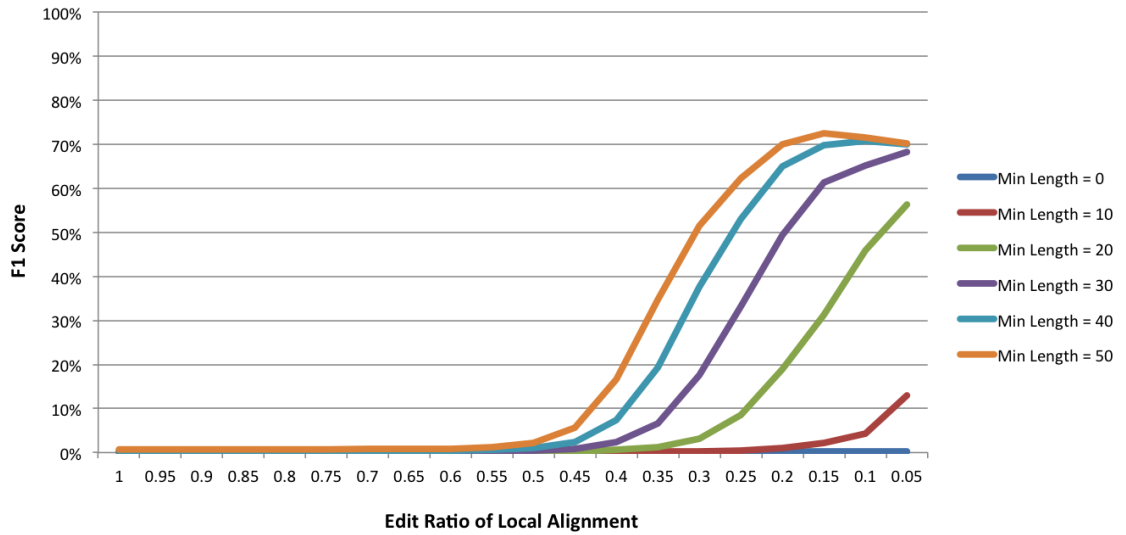


Figure A.13: F_1 scores of local alignment using PitchClass encoding for various alignment lengths in cover song identification on Lakh Covers dataset

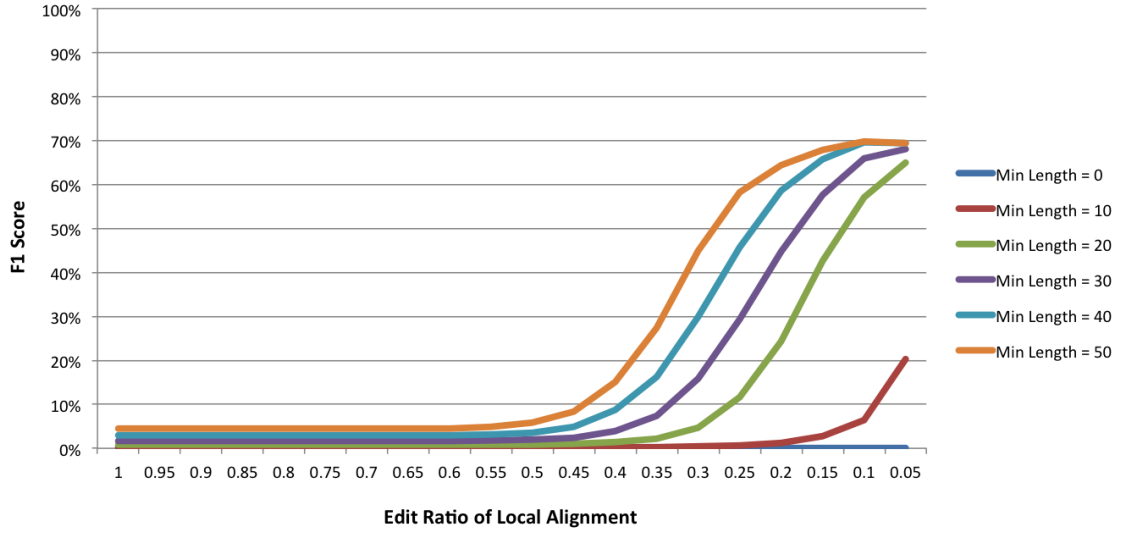


Figure A.14: F_1 scores of local alignment using Duration encoding for various alignment lengths in cover song identification on Lakh Covers dataset

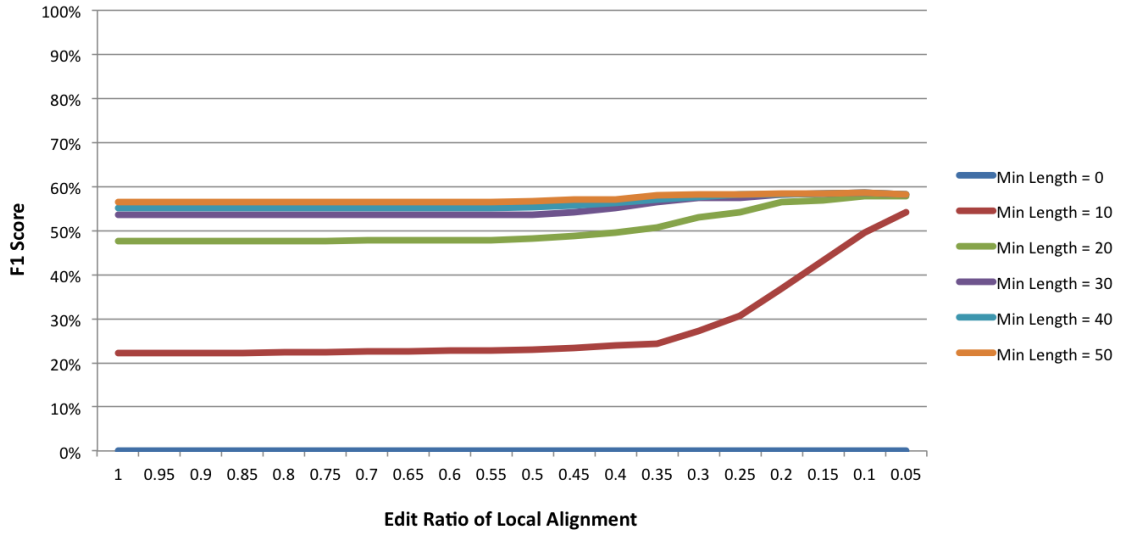


Figure A.15: F_1 scores of local alignment using PC+Duration encoding for various alignment lengths in cover song identification on Lakh Covers dataset

Appendix B

Cover Song Identification of Jazz Standard

Exact Matching

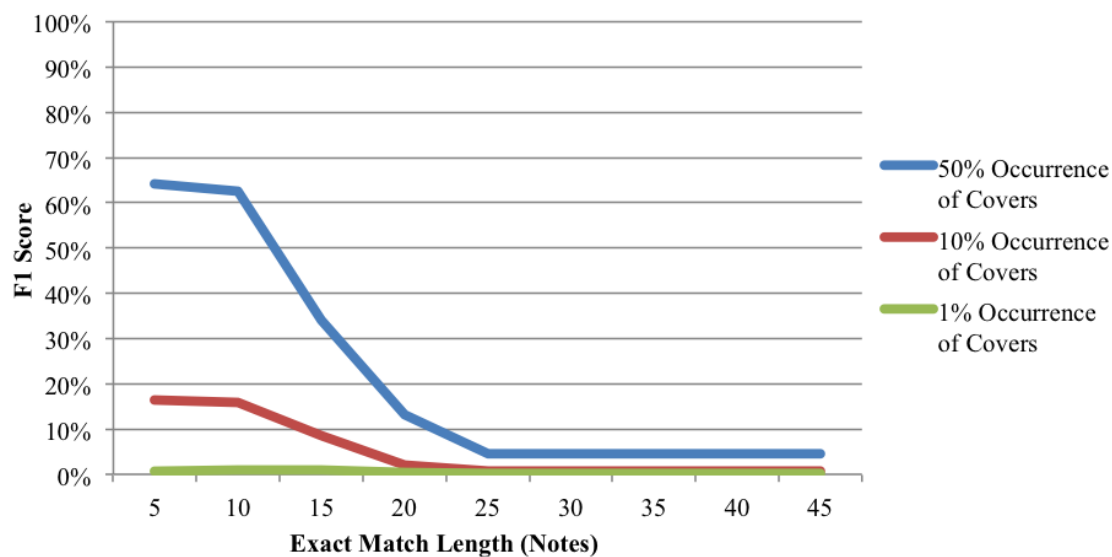


Figure B.1: F_1 scores of exact matching using Parsons encoding with different occurrences of covers in the Jazz Standard dataset

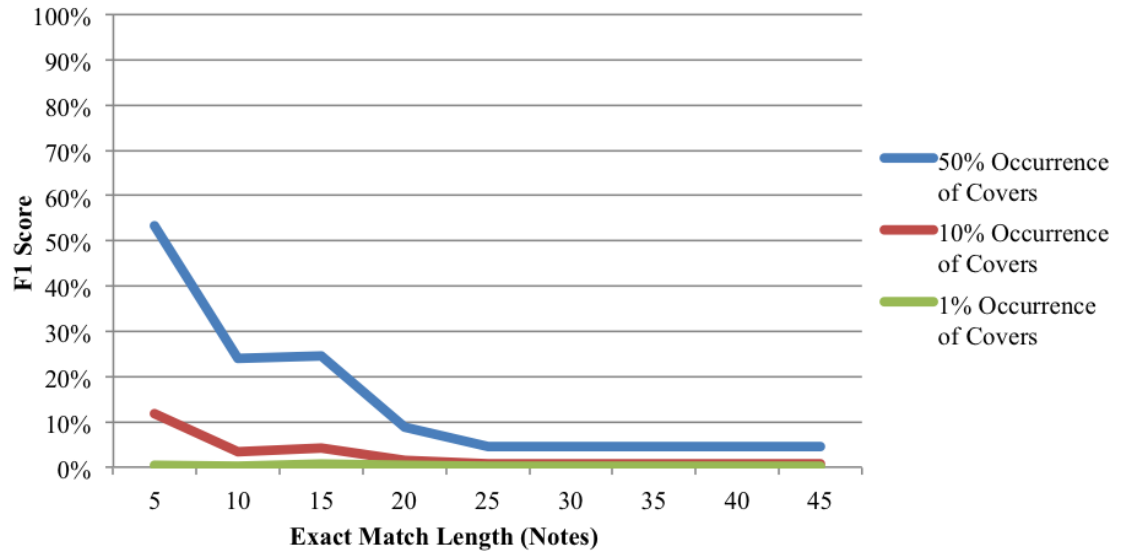


Figure B.2: F_1 scores of exact matching using Interval encoding with different occurrences of covers in the Jazz Standard dataset

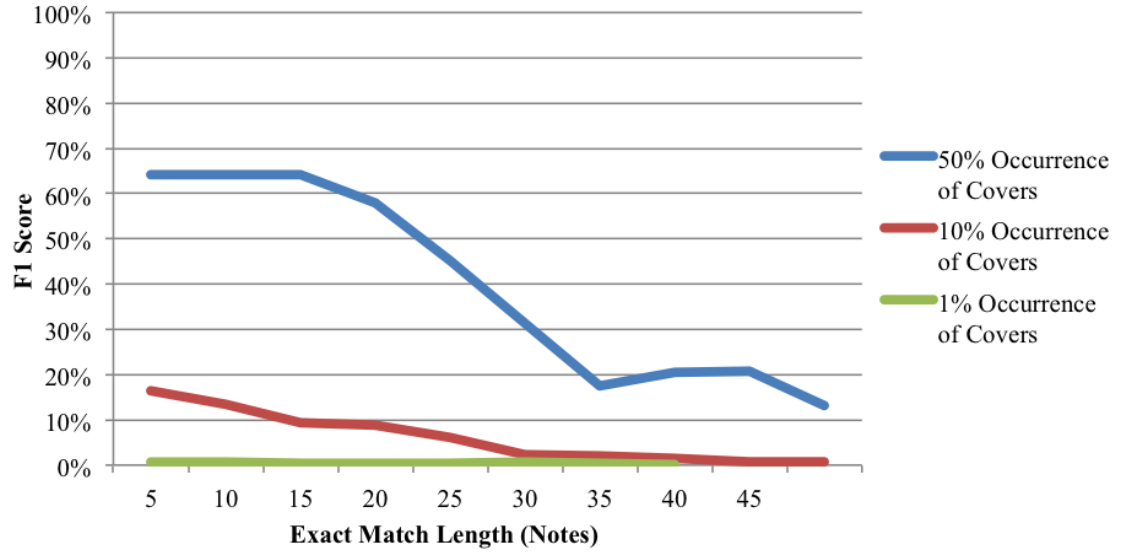


Figure B.3: F_1 scores of exact matching using Duration encoding with different occurrences of covers in the Jazz Standard dataset

Global Alignment

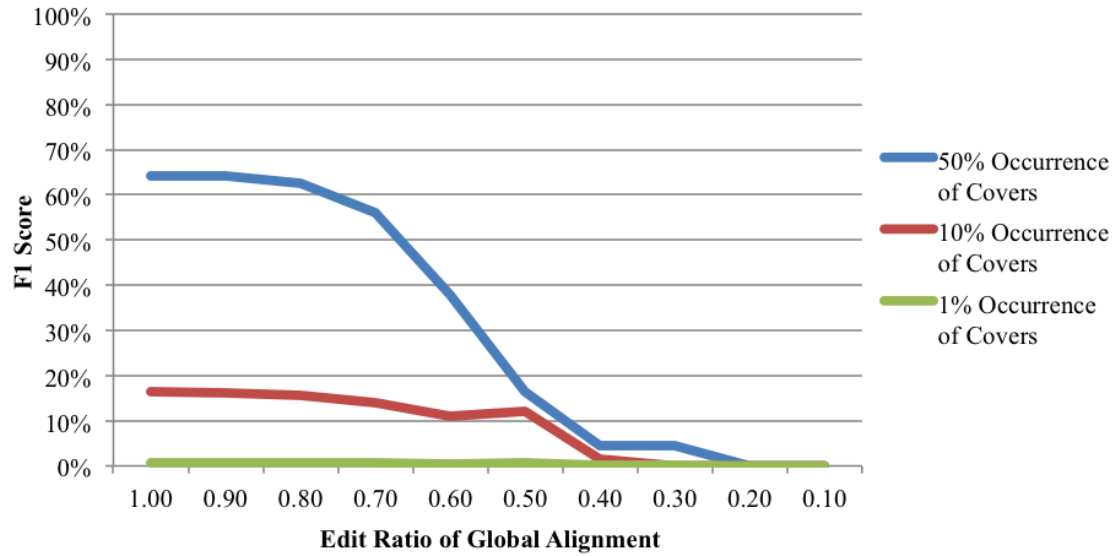


Figure B.4: F_1 scores of global alignment using Parsons encoding with different occurrences of covers in the Jazz Standard dataset

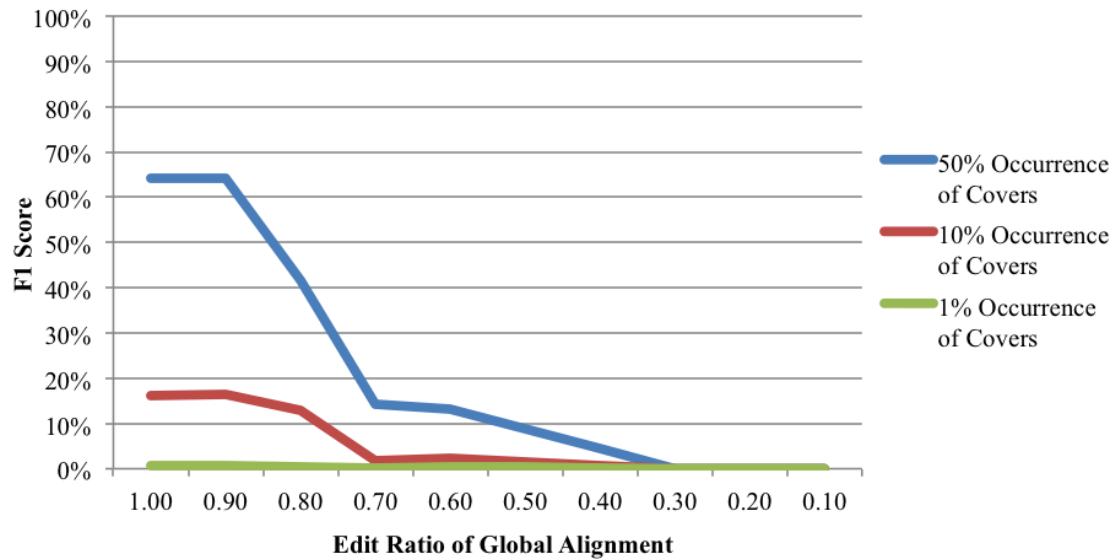


Figure B.5: F_1 scores of global alignment using Interval encoding with different occurrences of covers in the Jazz Standard dataset

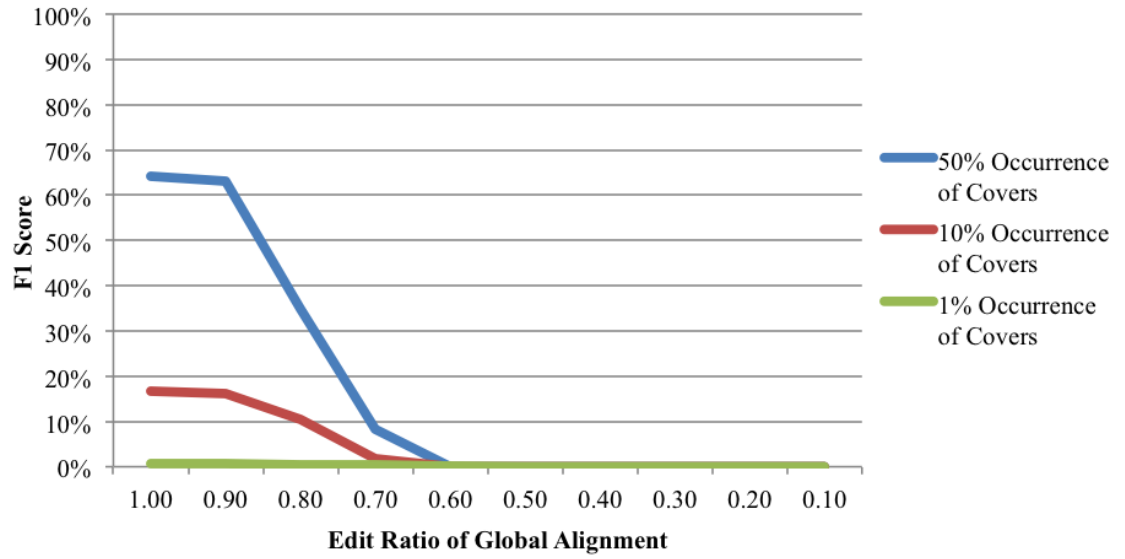


Figure B.6: F_1 scores of global alignment using Duration encoding with different occurrences of covers in the Jazz Standard dataset

Local Alignment

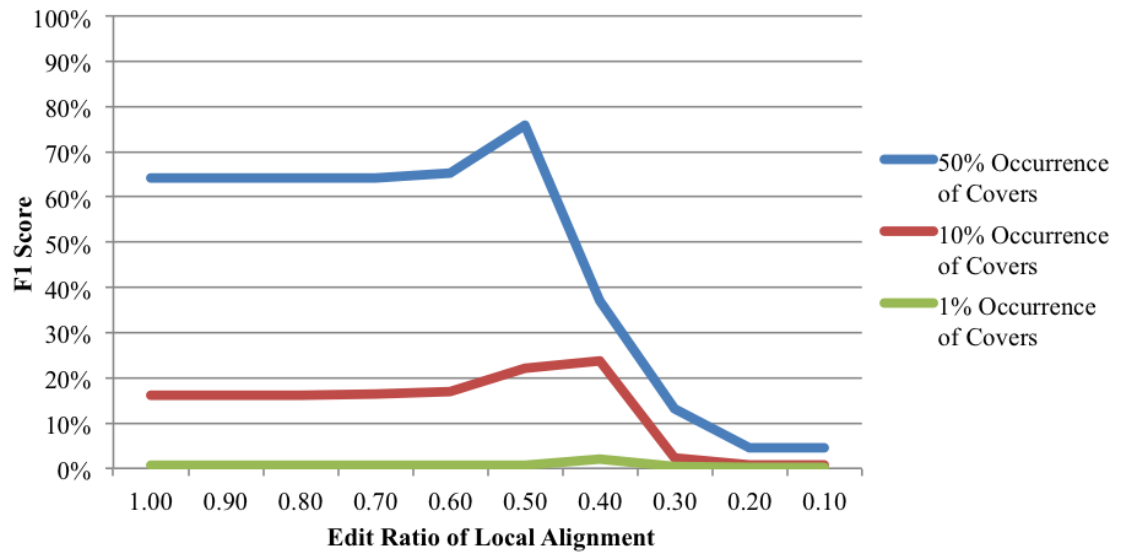


Figure B.7: F_1 scores of local alignment using Parsons encoding with different occurrences of covers in the Jazz Standard dataset

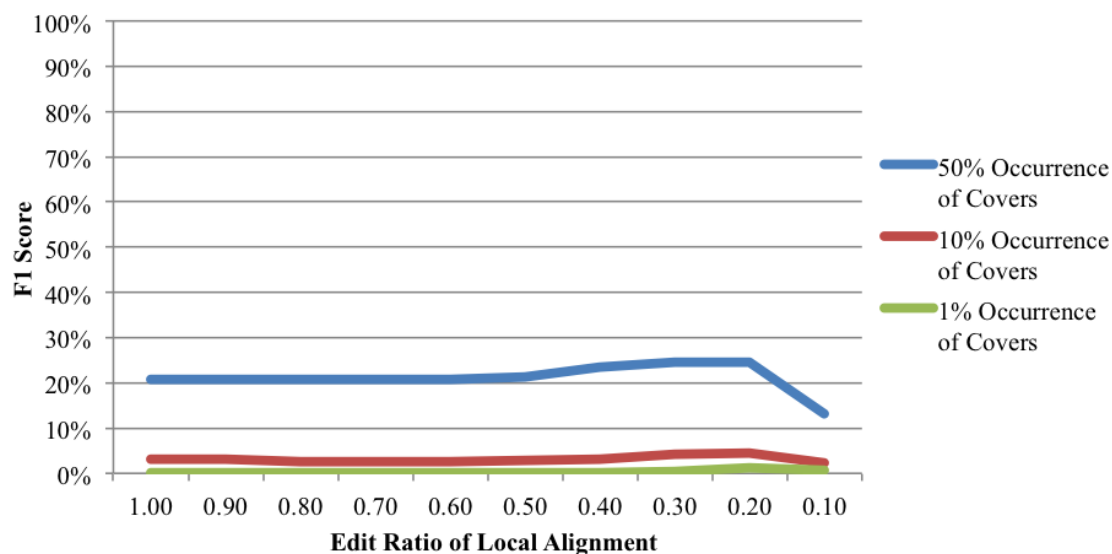


Figure B.8: F_1 scores of local alignment using Interval encoding with different occurrences of covers in the Jazz Standard dataset

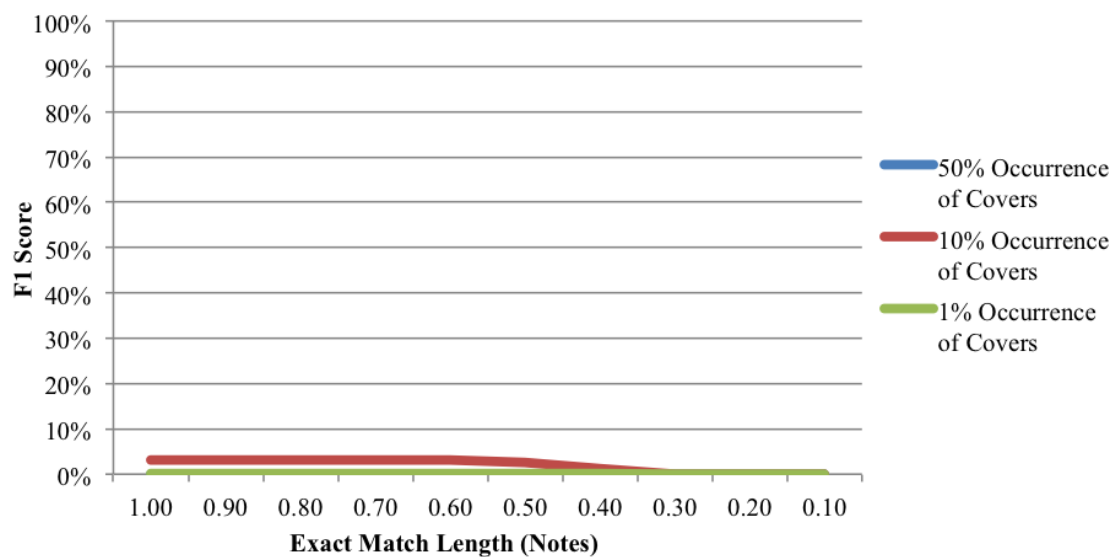


Figure B.9: F_1 scores of local alignment using Duration encoding with different occurrences of covers in the Jazz Standard dataset

References

- [1] E. Fetters, “The cambridge history of western music theory,” *Music Educators Journal*, vol. 93, no. 4, pp. 18–19, 2007.
- [2] R. Bagley, “The prehistory of chinese music theory,” in *Proceedings-British Academy*, vol. 131, p. 41, Oxford University Press Inc., 2005.
- [3] J. H. Chalmers, *Divisions of the Tetrachord: A Prolegomenon to the Construction of Musical Scales*. Frog Peak Music, 1993.
- [4] G. F. Scelta, “The history and evolution of the musical symbol,” *The History and Evolution of the Musical Symbol*, 2014.
- [5] A. Whittall, *The Cambridge Introduction to Serialism*. Cambridge University Press, 2008.
- [6] C. A. Tschider, “Automating music similarity analysis in ‘sound-alike’ copyright infringement cases,” *NYSBA Entertainment, Arts and Sports L. J.* 60, vol. Summer 2014, 2014.
- [7] D. R. Begault, H. D. Heise, and C. A. Peltier, “Analysis criteria for forensic musicology,” in *Proceedings of Meetings on Acoustics ICA2013*, vol. 19, p. 060005, ASA, 2013.
- [8] W. F. Patry, *Patry on Copyright*, vol. 1. Thompson West, 2007.
- [9] USLegal, “Substantial similarity law and legal definition,” 2017.
- [10] D. Bogdanov, J. Serrà, N. Wack, P. Herrera, and X. Serra, “Unifying low-level and high-level music similarity measures,” *IEEE Transactions on Multimedia*, vol. 13, no. 4, pp. 687–701, 2011.
- [11] G. Tzanetakis and P. Cook, “Musical genre classification of audio signals,” *IEEE Transactions on speech and audio processing*, vol. 10, no. 5, pp. 293–302, 2002.
- [12] M. Mckinney and J. Breebaart, “Features for audio and music classification,” in *Proceedings of the International Symposium on Music Information Retrieval*, pp. 151–158, 2003.

- [13] D. Li, I. K. Sethi, N. Dimitrova, and T. McGee, “Classification of general audio data for content-based retrieval,” *Pattern Recognition Letters*, vol. 22, no. 5, pp. 533–544, 2001.
- [14] B. Logan and A. Salomon, “A music similarity function based on signal analysis,” in *ICME*, pp. 22–25, 2001.
- [15] T. Li and M. Ogihara, “Content-based music similarity search and emotion detection,” in *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP’04)*, vol. 5, pp. V–705, IEEE, 2004.
- [16] E. Pampalk, *Computational Models of Music Similarity and their Application in Music Information Retrieval*. PhD thesis, Vienna University of Technology, Vienna, Austria, March 2006.
- [17] F. Zheng, G. Zhang, and Z. Song, “Comparison of different implementations of MFCC,” *Journal of Computer Science and Technology*, vol. 16, no. 6, pp. 582–589, 2001.
- [18] S. S. Stevens, J. Volkman, and E. B. Newman, “A scale for the measurement of the psychological magnitude pitch,” *The Journal of the Acoustical Society of America*, vol. 8, no. 3, pp. 185–190, 1937.
- [19] A. M. Noll, “Short-time spectrum and “cepstrum” techniques for vocal-pitch detection,” *The Journal of the Acoustical Society of America*, vol. 36, no. 2, pp. 296–302, 1964.
- [20] V. Velardo, M. Vallati, and S. Jan, “Symbolic melodic similarity: State of the art and future challenges,” *Computer Music Journal*, vol. 40, no. 2, pp. 70–83, 2016.
- [21] D. Jurafsky and J. H. Martin, *Speech and Language Processing*, vol. 3. Pearson, 2014.
- [22] R. A. Wagner and M. J. Fischer, “The string-to-string correction problem,” *Journal of the ACM (JACM)*, vol. 21, no. 1, pp. 168–173, 1974.
- [23] T. Miura and I. Shioya, “Similarity among melodies for music information retrieval,” in *Proceedings of the Twelfth International Conference on Information and Knowledge Management*, pp. 61–68, ACM, 2003.
- [24] H. H. Hoos, K. Renz, and M. Görg, “GUIDO/MIR - an experimental musical information retrieval system based on GUIDO music notation,” in *ISMIR*, pp. 41–50, 2001.

- [25] E. Pollastri and G. Simoncelli, “Classification of melodies by composer with hidden Markov models,” in *Proceedings. First International Conference on Web Delivering of Music*, pp. 88–95, IEEE, 2001.
- [26] G. Aloupis, T. Fevens, S. Langerman, T. Matsui, A. Mesa, Y. Nuñez, D. Rapaport, and G. Toussaint, “Algorithms for computing geometric measures of melodic similarity,” *Computer Music Journal*, vol. 30, no. 3, pp. 67–76, 2006.
- [27] Y. Zhu, M. Kankanhalli, and Q. Tian, “Similarity matching of continuous melody contours for humming querying of melody databases,” in *Multimedia Signal Processing, 2002 IEEE Workshop on*, pp. 249–252, IEEE, 2002.
- [28] L. Hofman-Engl, *Melodic Transformations and Similarity: A Theoretical and Empirical Approach*. PhD thesis, Phd Thesis, Keele University, 2002.
- [29] A. Ghias, J. Logan, D. Chamberlin, and B. C. Smith, “Query by humming: Musical information retrieval in an audio database,” in *Proceedings of the Third ACM International Conference on Multimedia*, pp. 231–236, ACM, 1995.
- [30] M. A. Schmuckler, “Testing models of melodic contour similarity,” *Music Perception: An Interdisciplinary Journal*, vol. 16, no. 3, pp. 295–326, 1999.
- [31] W. Jeon, C. Ma, and Y. M. Cheng, “An efficient signal-matching approach to melody indexing and search using continuous pitch contours and wavelets,” in *ISMIR*, pp. 681–686, 2009.
- [32] R. Tushar, “Longest common substring,” March 2015. <https://www.youtube.com/watch?v=BysNXJHzCEs> Accessed: 2017-01-30.
- [33] “Suffix trees in computational biology.” http://homepage.usask.ca/~ctl271/857/suffix_tree.shtml. Accessed: 2017-01-29.
- [34] R. Giegerich and S. Kurtz, “From Ukkonen to McCreight and Weiner: A unifying view of linear-time suffix tree construction,” *Algorithmica*, vol. 19, no. 3, pp. 331–353, 1997.
- [35] B. Langmead, “Tries and suffix tries,” Fall 2013. http://www.cs.jhu.edu/~langmea/resources/lecture_notes/tries_and_suffix_tries.pdf.
- [36] D. Gusfield, “Linear-time construction of suffix trees,” *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*, 1997.
- [37] U. Manber and G. Myers, “Suffix arrays: A new method for on-line string searches,” *siam Journal on Computing*, vol. 22, no. 5, pp. 935–948, 1993.

- [38] G. W. Klau, “Suffix trees and suffix arrays,” 2005. http://www.inf.fu-berlin.de/lehre/WS05/aldabi/downloads/stringMatching_part2.pdf.
- [39] S. Burkhardt and J. Kärkkäinen, “Fast lightweight suffix array construction and checking,” in *Annual Symposium on Combinatorial Pattern Matching*, pp. 55–69, Springer, 2003.
- [40] M. Gollery, “Bioinformatics: Sequence and genome analysis,” *Clinical Chemistry*, vol. 51, no. 11, pp. 2219–2220, 2005.
- [41] S. B. Needleman and C. D. Wunsch, “A general method applicable to the search for similarities in the amino acid sequence of two proteins,” *Journal of Molecular Biology*, vol. 48, no. 3, pp. 443–453, 1970.
- [42] T. F. Smith and M. S. Waterman, “Comparison of biosequences,” *Advances in Applied Mathematics*, vol. 2, no. 4, pp. 482–489, 1981.
- [43] Y.-L. Lo, W.-L. Lee, and L.-h. Chang, “True suffix tree approach for discovering non-trivial repeating patterns in a music object,” *Multimedia Tools and Applications*, vol. 37, no. 2, pp. 169–187, 2008.
- [44] T.-C. Chou, A. L. Chen, and C.-C. Liu, “Music databases: Indexing techniques and implementation,” in *Proceedings of International Workshop on Multimedia Database Management Systems*, pp. 46–53, IEEE, 1996.
- [45] A. L. Chen, M. Chang, J. Chen, J.-L. Hsu, C.-H. Hsu, and S. Y. Hua, “Query by music segments: An efficient approach for song retrieval,” in *Multimedia and Expo, 2000. ICME 2000. 2000 IEEE International Conference on*, vol. 2, pp. 873–876, IEEE, 2000.
- [46] M. Jekovec, J. Demsar, and A. Brodnik, “Computer aided melodic analysis using suffix tree,” in *ICMC*, 2012.
- [47] O. Lartillot, S. Dubnov, G. Assayag, and G. Bejerano, “Automatic modeling of musical style,” in *ICMC*, 2001.
- [48] J. P. Bello, “Audio-based cover song retrieval using approximate chord sequences: Testing shifts, gaps, swaps and beats,” in *ISMIR*, vol. 7, pp. 239–244, Citeseer, 2007.
- [49] P. Ferraro and P. Hanna, “Optimizations of local edition for evaluating similarity between monophonic musical sequences,” in *Large Scale Semantic Access to Content (Text, Image, Video, and Sound)*, pp. 64–69, Le Centre de Hautes Études Internationales d’Informatique Documentaire, 2007.

- [50] J. D. Frey, “Finding song melody similarities using a DNA string matching algorithm,” Master’s thesis, Kent State University, 2008.
- [51] J. Urbano, J. Lloréns, J. Morato, and S. Sánchez-Cuadrado, “Mirex 2010 symbolic melodic similarity: Local alignment with geometric representations,” *Music Information Retrieval Evaluation eXchange*, 2010.
- [52] F. Gómez, A. Pikrakis, J. Mora, J. M. Díaz-Báñez, E. Gómez, and F. Escobar, “Automatic detection of ornamentation in flamenco,” in *Fourth International Workshop on Machine Learning and Music MML*, 2011.
- [53] C. Raffel, *Learning-Based Methods for Comparing Sequences, with Applications to Audio-to-MIDI Alignment and Matching*. PhD thesis, Columbia University, 2016.
- [54] D. Temperley, “What’s key for key? the Krumhansl-Schmuckler key-finding algorithm reconsidered,” *Music Perception: An Interdisciplinary Journal*, vol. 17, no. 1, pp. 65–100, 1999.
- [55] R. Hart, “Key-finding algorithm.” <http://rnhart.net/articles/key-finding/>. Accessed: 2016-12-14.
- [56] C. L. Krumhansl and E. J. Kessler, “Tracing the dynamic changes in perceived tonal organization in a spatial representation of musical keys,” *Psychological Review*, vol. 89, no. 4, p. 334, 1982.
- [57] C. S. Sapp, J. O. Smith, C. Chafe, and E. Selfridge-Field, *Computational Methods for the Analysis of Musical Structure*. Stanford University, 2011.
- [58] J. Kryzanowska, “Intro to white mensural notation.” http://blowthyhorn.com/blog/wp-content/uploads/2015/07/white_mensural_notation.pdf, 2013. Accessed: 2017-05-23.
- [59] C. Manning, “Evaluation of text classification: Precision, recall, and the F measure.” <https://www.youtube.com/watch?v=X0gvlpwu0hI>, 2012. Accessed: 2017-05-25.
- [60] D. C. Blair, *Information Retrieval*. Wiley Online Library, 1979.
- [61] M. S. Rosenberg, “Evolutionary distance estimation and fidelity of pair wise sequence alignment,” *Bmc Bioinformatics*, vol. 6, no. 1, p. 102, 2005.
- [62] L. Benuskova, “Cosc 348: Computing for bioinformatics.” <http://www.cs.otago.ac.nz/cosc348/>.

- [63] S. Henikoff and J. G. Henikoff, “Amino acid substitution matrices from protein blocks,” *Proceedings of the National Academy of Sciences*, vol. 89, no. 22, pp. 10915–10919, 1992.