# Correlational Harmonic Metrics: Bridging Computational and Human Notions of Musical Harmony

# Correlational Harmonic Metrics:

# Bridging Computational and Human Notions of Musical Harmony

*A thesis presented by*

Dylan Freedman

submitted in partial fulfillment of the requirements for an AB degree

with honors in Computer Science and Music

Departments of Computer Science and Music

Harvard College

April 1, 2015

# Abstract

The goal of this paper is to show that traditional music information retrieval tasks with well-chosen parameters perform similarly using computationally extracted chord annotations versus ground-truth annotations. Using a collection of Billboard songs from the last 60 years with provided ground-truth chord labels, I use established automatic chord identification algorithms to produce a corresponding extracted chord label dataset. I devise methods to compare chord progressions between two songs on the basis of their optimal localized alignment scores, adapting traditional sequence alignment techniques for transposition-invariance and the chord alphabet. I create a set of chord progression comparison parameters defined by chord distance metrics, gap costs, and normalization measures and run a black-box global optimization algorithm to stochastically search for the best parameter set to perform chordal comparisons on collections of songs across two primary tasks—*fully connected harmonic comparison* and *query by n-grams*. The first task involves evaluating chord progression similarity between all pairwise combinations of songs, separately ranking results for ground-truth and extracted chord labels, and returning the Spearman rho rank correlation coefficient of the two resulting rankings. The second task harmonically compares random chord query sequences of different sizes to the songs in the datasets, for each query ranking results for ground-truth and extracted chord labels, and returning the average Spearman rho rank correlation coefficient of all pairs of resulting rankings. These methods reflect common harmonic music information retrieval objectives and are robust and rapid, performing more efficiently than existing chord sequence alignment methods and introducing the use of correlational harmonic metrics between collections of songs.

# Acknowledgements

I would like to acknowledge first and foremost my thesis advisors, Professors Eddie Kohler and Hans Tutschku, for their helpful advice and insights. I extend gratification to the Music Information Retrieval community for promoting with active fervor an intersection of fields that inspired my research. I am deeply thankful for the educational opportunities that have afforded me the serendipity to fall into such tracks of learning, along with friends from the past four years who have offered their own invaluable contributions. I am wholly indebted in support and heartfelt appreciation to my eccentric family.

# Contents

# List of Figures

All figures copyright of the author.

# Introduction

The word "harmony" in its Greek origin, *Armonia*, represents an ordering and balance [23], a term whose meaning has transformed and over-accommodated through the musical eras of expression [47]. Harmony is a combination of distinctly different *concords* and *discords*, pleasing and unpleasing sounds to some theorists [3]; others hear these pleasant and clashing sounds as phenomena describing one and the same[1] [40, 41]. Harmony is rooted in a singular note and its natural, physical overtone series to purists [14]; the vertical combinations of stacked notes to traditionalists [3]; and the cultural contexts of a chord to musicologists [42]. It is to some a sensation [46] and others a rigid set of rules [37, 43]. It has materiality and simultaneously only exists in the abstract [47]. To one, harmony *is* love, the genealogical father of melody [14], and to another, harmony constitutes a multicellular biological organism[2] [39].

This paper strives to capture an ambiguous, human, aesthetical notion and faithfully preserve its abstraction as a computational metric for comparison. While in no way trying to resolve a centuries-old debate on what exactly harmony *is* or how it should be labeled, this paper instead asks how different understandings of harmony—human and computer—can be reconciled, and how works of music can be harmonically compared in spite of these diverging notions. Within this paper, harmony is extracted from pieces of music in the form of *chord symbols*—labels which describe sequential, nonoverlapping vignettes into a song's vertical combinations of notes. Though describing a complex musical system in terms of a sequence of concise qualitative descriptors is limiting, this representation is not arbitrary as chord symbols are standardly used. Additionally, the reduction of harmonic content to chord symbols provides a depth of detail neither too verbose nor too succinct, ripe for computational comparison. Chord symbols, devoid of any duration, tempo, dynamic,

[1]Arnold Schönberg, modern theorist and philosopher, conceives of an *emancipation of dissonance* in which consonance and dissonance are one and the same as they both describe the relation of a note and its overtones. Consonances are based on more proximal and thus audible overtones, whereas dissonances describe more remote overtones.

[2]Theorist Heinrich Schenker draws a musical parallel to biology, comparing an individual note to a single cell with the natural instinct to reproduce, propagating motifs, harmonies, and complete works of music.

instrumentation, or timbre, match an intuitive manner in which people discuss harmonic similarity, in terms of locally and globally matching regions of labels that describe chordal semantics between songs [12].

Existing chord recognition algorithms can convert an audio file into chord labels—a process called *extraction*—with an accuracy of up to 80% correct identification per chord compared with human annotation [21, 28]. Though, philosophically, a singular *right* conception of harmony against which to reference extraction algorithms is impossible to establish, researchers have released extensively verified human-annotated chord label datasets for popular Western songs from the 1950s to the 1990s by hiring musical experts and double-checking all harmonic transcriptions [6]. The difficulty of establishing *ground-truth*, or manually verified, reference data in the realm of musical harmony [10] has not suppressed the burgeoning field of *music information retrieval* (MIR), which draws on music theory, psychology, signal processing, and artificial intelligence to extract information from music. The Music Information Retrieval Evaluation eXchange[3] (MIREX), an international symposium for computational music recognition tasks, includes a chord labelling contest tested against the reference dataset in [6] and others [18, 31] that has enticed a growing research interest [21, 28, 29, 36, 49].

Chord recognition algorithms provide a tool for converting an audio file into a series of chord labels representative of the computational understanding of that piece's harmony. Once this notion of harmony is extracted, two pieces of music can be computationally compared through the use of *metrics* of similarity that explicitly quantify subjective, qualitative harmonic similarity judgments. There are metrics for comparing two individual chord labels [18, 26, 38] and metrics for comparing entire progressions of chords which usually rely on subcalculations of the former [2, 10, 11, 16, 17]. These metrics are detailed in depth in chapter 2. While there is some notion of human-generated ground-truth data for chord extraction, there is no reference dataset or MIREX competition for harmonic progression similarity [10]—the desire for higher level similarity metrics motivates the work of this paper.

Many different MIR tasks rely on metrics of harmonic progression similarity, including harmonic content querying [1], cover song identification [20], and automated harmonic analysis [4]. This paper fundamentally asks how basic MIR tasks' results compare between the use of computationally extracted chord data and human-annotated ground-truth data, leading to my creation of a similarity measure at a higher level of MIR understanding: *correlational metrics.* These metrics quantify how similarly a MIR task performs using

[3]`http://www.music-ir.org/mirex/wiki/MIREX_HOME`

datasets that reflect differing harmonic understandings. Maximizing a correlational harmonic metric corresponds to optimizing the parameters of a MIR algorithm such that different input datasets yield relatively similar results. While it could be argued that such a metric only encourages similar results rather than *good* results, the experiments in this paper use established effective algorithms to analyze a significantly large-scale amount of harmonic data. Using correlational metrics, this paper demonstrates established harmonic MIR algorithms operate productively with large quantities of extracted chord data, resolving the debate on an exact notion of harmony in massive music informatics applications for a practical computational notion.

This paper begins with an overview of musical harmony and terms in chapter 1, without delving into historical accounts of rules of harmonic progression or placing subjective judgments of what sounds *good* or *bad*, concordant or discordant. I introduce basic computer algorithms such as the *Fourier transform* and visual representations of audio content, emphasizing the inaccuracy inherent in computational extraction at the note level and the ambiguity which precludes any possibility of perfect extraction. Readers acquainted with this background on the intersection of music with computation and its briefly discussed physical and psychological implications are welcome to skip this chapter. Chapter 2 discusses computational metrics for harmonic comparison, including existing techniques and novel explorations, with discussion of their advantages and disadvantages for harmonically comparing two songs according to outlined subjective principles. Chapter 3 rationalizes the overall design decisions used to blueprint the experimentation and establishes the primary tasks in the paper. Chapter 4 elaborates on the implementation mechanics and code used to both accomplish the objectives and set up the experimentation. Finally, chapter 5 details experimental results and is followed by a conclusion and overall discussion in chapter 6.

Chapter 1

# Overview of Music and Computation

## 1.1 A Primer on Western Music Theory

### 1.1.1 Notes: The Basic Building Block

In music, a *note* is the most basic element. A note is based on pitch, a subjective and perceptual property. Though the pitch of a note is closely related and usually resembles its objective physical frequency (as measured in Hertz, or cycles per second, of a waveform), pitch differs in that its semantic meaning is derived from the listener. This distinction can be demonstrated with a visual analogy used by Terheardt [48] in Figure 1.1 in which the word `PITCH` is apparent even though the visual information suggests only shadow – a pitch can be heard even if its perceived frequency is not physically present. A note also consists of a duration.

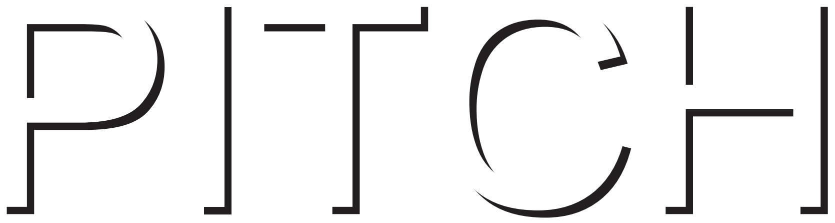

Figure 1.1: Terheardt's visual pitch analogy. In this illusion, the eye perceives contours not present. Pitch describes the information received by a listener even if physical frequencies are not present.

Western music is based on a division of 12 distinct frequencies per *octave*. An octave is an *interval*, or distance between two frequencies, that corresponds to a power of 2 multiplication. Musical pitch is perceived in a logarithmic scale—one octave above a given perceived frequency is double that frequency; one octave below is half that frequency. The progression of notes containing all 12 pitches in succession in an octave is called a *chromatic scale*. A *semitone*, or *half-step*, is the smallest interval, equal to 1/12 of an octave. $n$ semitones above a given frequency $f_0$ or $-n$ below can be calculated as $f_0 \cdot 2^{n/12}$.

*Note names* are used to classify the pitches in the chromatic scale. Note names consist of a base name and 0 or more *accidentals*. The base names of a note correspond to the white keys on a piano—in any one given octave there are the following names: $C$, $D$, $E$, $F$, $G$, $A$, and $B$. A base note name can optionally be decorated with an indefinite number of sharps (♯) or flats (♭), but not both, in the note name. This can be illustrated with the following grammar[1] (figure 1.2):

$$\begin{aligned}
NoteName &\rightarrow BaseNote \mid BaseNote\ SharpAccidentals \mid BaseNote\ FlatAccidentals \\
BaseNote &\rightarrow \mathbf{C} \mid \mathbf{D} \mid \mathbf{E} \mid \mathbf{F} \mid \mathbf{G} \mid \mathbf{A} \mid \mathbf{B} \\
SharpAccidentals &\rightarrow \#\ SharpAccidentals \mid \# \\
FlatAccidentals &\rightarrow \boldsymbol{b}\ FlatAccidentals \mid \boldsymbol{b}
\end{aligned}$$

Figure 1.2: Context-free grammar of a note name

Sharps and flats are referred to as accidentals. Each additional (#) increases the pitch to which the note name refers by 1 semitone; likewise, each ($b$) decreases the pitch by 1 semitone. The black keys on the piano represent pitches 1 semitone in between the surrounding white keys. Each white key is either 1 semitone or 2 semitones apart, depending on if a black key is in the middle. For instance, $C$ and $D$ are 2 semitones apart since there is a black key in between them, whereas $E$ and $F$ are 1 semitone apart. See Figure 1.3 for a visual of the piano with note names and Figure 1.6 for a diagram depicting intervals on the piano.

The divergence in musical notes—one of the most basic elements—between exactly calculable and subjectively derived from perception gives rise to many questions in *music informatics retrieval* (MIR), a field devoted to automatically extracting data and classifying features from works of music.

[1] A grammar describes production rules that recursively generate acceptable symbol sequences in a language. In the language of musical notes, these rules structurally outline the syntax of a well-formed note name.

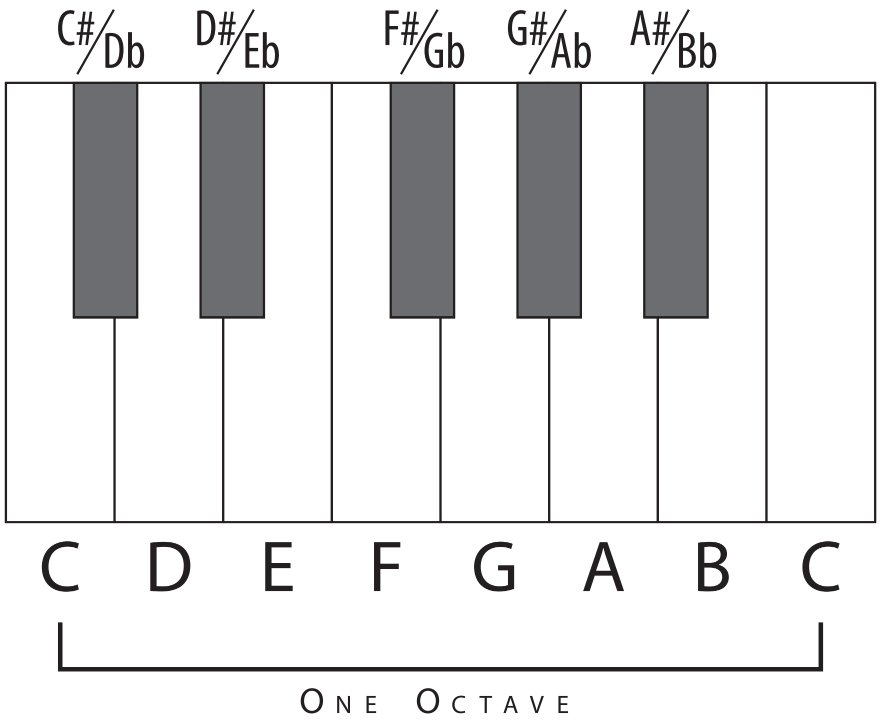

Figure 1.3: One octave on a piano

### 1.1.2 Pitch Class

Though the chromatic scale contains 12 notes, there are an infinite amount of ways to represent any singular pitch within the octave. For instance, a $C\sharp$ pitch can be represented as a $D\flat$, a $B\sharp\sharp$ ("B-double-sharp"), or a $F\flat\flat\flat\flat$, among other possibilities. There are notational reasons to represent a pitch in these ways; outside of *equal temperament*, the tuning system upon which Western music is based, these notes sound distinct and have different perceptual frequencies. In equal temperament, which is an assumption guiding this paper, all these different representations of the same chromatic note have identical pitches.

A *pitch class* is the collection of all identical pitches across all octaves. The $C\sharp$ pitch class, for instance, contains all the $C\sharp$ pitches over all the octaves, the $D\flat$ pitches over all the octaves, and any other pitch that represents the same chromatic note across all octaves.

### 1.1.3 Chords and Harmonies

A chord consists of a combination of notes sounding simultaneously or close enough in succession to resemble a texture. The *Harvard Dictionary of Music* defines a chord as consisting of at least three notes [3]. A chord perceptually describes the notes that are contained within.

Chords are commonly labeled with qualities, which describe the intervals between the pitch classes involved, invariant of octave. Notes can be replicated across octaves as long as they occur at least once, and the ordering can be changed. Different orderings and octave choices in a chord are called *voicings.*

A *major* chord consists of a *root note*, the base pitch class from which successive intervals are constructed, and pitch classes 4 semitones and 7 semitones above the root note modulus 12. This can be notated as a list of intervals, $0, +4, +7$ mod 12, but for convenience, the root note corresponding to interval 0 can be omitted. See Figure 1.4 for a sample of commonly named chord qualities and the associated intervals.

| Chord Quality | Shorthand | Intervals from Root (mod 12) |
|---|---|---|
| Major | | +4, +7 |
| Major 6th | 6 | +4, +7, +8 |
| Major 7th | maj7 | +4, +7, +11 |
| Minor | m | +3, +7 |
| Minor 6th | m6 | +3, +7, +9 |
| Minor 7th | m7 | +3, +7, +10 |
| Dominant 7th | 7 | +4, +7, +10 |
| Augmented | aug | +4, +8 |
| Diminished | dim | +3, +6 |
| Diminished 7th | dim7 | +3, +6, +9 |
| Half-diminished 7th | m7b5 | +3, +6, +10 |

Figure 1.4: Common Chord Qualities and Associated Intervals

Chords are labeled with their root note followed by their quality, like *Eb* minor, *B* augmented, or *F* half-diminished 7th. A chord with only 3 notes in which successive intervals are within an octave from the root note is called a *triad.* A *C* major triad is demonstrated in Figure 1.5.

The *bass* note of a chord is its lowest note. A chord's bass note is often its root, however this is not always the case depending on voicing. When the notes of a chord are such that its root note is not the bass note, that chord is said to be *inverted.* When the bass note of a chord is not the chord's root nor any of the

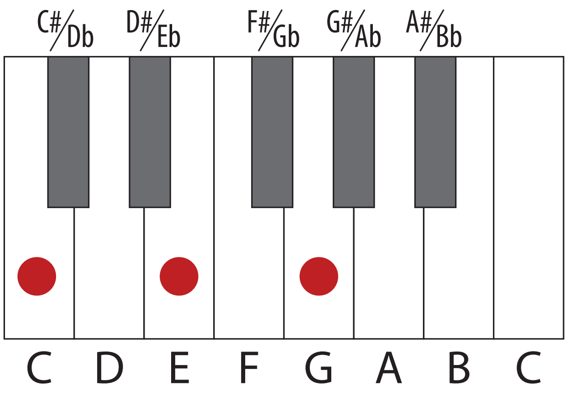

Figure 1.5: A $C$ major triad on the piano. The notes indicated with the red circle are $C$, the first circle on the left; $E$, 4 semitones above $C$; and $G$, 7 semitones above $C$.

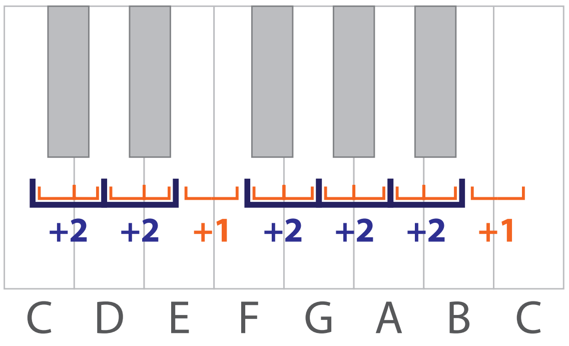

Figure 1.6: The intervals in semitones between successive white keys on the piano. For instance, $C$ to $E$ are $(2 + 2) = 4$ semitones apart; $C$ and $G$ are $(2 + 2 + 1 + 2) = 7$ semitones apart.

pitches involved within the chord's quality, that chord is called a *slash chord*. The name slash chord refers to its notation—a $D$ minor chord with a bass of $B$ is notated as $Dm/B$.

The notation of a chord can be outlined with the following context-free grammar:

$$\begin{aligned}
Chord &\rightarrow Root\ Quality \mid Root\ Quality\ /\ Bass \\
Root &\rightarrow NoteName \\
Bass &\rightarrow NoteName \\
Quality &\rightarrow \text{maj} \mid 6 \mid \text{maj7} \mid \text{m} \mid \text{m6} \mid \text{m7} \mid 7 \mid \text{aug} \mid \text{dim} \mid \text{dim7} \mid \text{m7b5}
\end{aligned}$$

where $NoteName$ is a note name according to the context-free grammar in Figure 1.2.

### 1.1.4 Key Signature

The *key signature* of a song describes the *tonic* or base harmony against which other chords perceptually resolve. Key signatures consist of a root note and a *mode*—major or minor—which describe the mood of the piece and outline the expected chords and pitch classes used in the song.

The $C$ major key signature consists of all the white keys on the piano. The pitch classes of notes in a song in C major are expected to fall on these keys. Only certain chords consist of pitch classes that are in

$C$ major. Starting with the note $C$ and ascending upwards in triad chords the following list of harmonies is obtained: $(C, Dm, Em, F, G, Am, Bdim)$.

It can be useful to construct a template of acceptable pitch classes in C major. All the notes in a C major key signature starting at $C$ are $(C, D,\ E,\ F,\ G,\ A,\ B)$. In terms of intervals of each of these notes relative to the previous, starting at the second element $(D)$, this list can be written $(+2, +2, +1, +2, +2, +2)$. With this template, it can be easy to obtain a list of acceptable pitch classes in other key signatures. For instance, the pitch classes $F\#$ major can be calculated by adding each interval to the last played note starting at $F\#$ obtaining $(F\#, G\#, A\#, B, C\#, D\#, F)$. The $A$ minor mode consists of all the white keys on the piano starting at $A$, and thus its template can be described $(+2, +1, +2, +2, +1, +2)$. See Figure 1.7 for a key signature template table.

| Key Signature Mode | Interval Template (mod 12) |
|---|---|
| Major | +2, +2, +1, +2, +2, +2 |
| Minor | +2, +1, +2, +2, +1, +2 |

Figure 1.7: The intervals from the previous note starting at the root that can be used to construct all the acceptable pitch classes in a key signature given the mode.

### 1.1.5 Chord Progressions

*Chord progressions* are sequences of chords in a song. There is a degree of subjectivity in identifying progressions of chords as chords can be overlapping, contain notes that straddle and linger between chords, or involve unknown or difficult to identify chord qualities. There are often perceptually obvious answers to what chords are playing based on musical cues ingrained in culture, commonalities across songs, and more advanced techniques in music theory.

Chord progressions are frequently notated with dashes in between, such as $Cmaj7 - Dm - G7 - Cmaj7$. Chord progressions are regarded as identical even if they are *transposed*, or the pitch class of the root and bass notes of each chord is shifted a certain amount. For instance, $Dmaj7 - Em - A7 - Dmaj7$ describes an identical progression as each chord is transposed up 2 semitones. A transposition-invariant representation is useful for classifying relatively similar chord progressions. Music theorists typically use roman numerals to represent the root and bass notes of the chord relative to the key of the song.

The roman numeral I corresponds to the root of the key signature, and successive notes are represented by successive roman numerals. The chord progression $Cmaj7 - Dm - G7 - Cmaj7$ could be rewritten as $\mathrm{I}maj7 - \mathrm{ii} - \mathrm{V}7 - \mathrm{I}maj7$ in the key of $C$ major. It is important to note that *minor* chords, or chords with qualities that are described as minor, are represented with lower case numerals, and the normal minor chord representation $(m)$ can be omitted.

## 1.2 Audio Files

### 1.2.1 Storing Audio

Music is stored digitally as a series of *amplitudes.* Amplitudes capture the energy in the compressions and rarefactions in the air that give sound to everything one can hear. Periodic fluctuations in amplitude represent the frequencies which give the listener a perceptual understanding of pitch. Digital storage of music invariably loses some information through *quantization*, the process of making the continuous data sound waves represent into discrete values computers can comprehend. As a visual analogy, in order for a picture to be rendered on a computer screen it needs to fit into the rectangular shape of pixels (see Figure 1.8)—likewise, audio needs to fit into discrete bins of amplitude values, thus forfeiting a minute amount of information. The number of bins per second is referred to as the *sample rate* of a song.

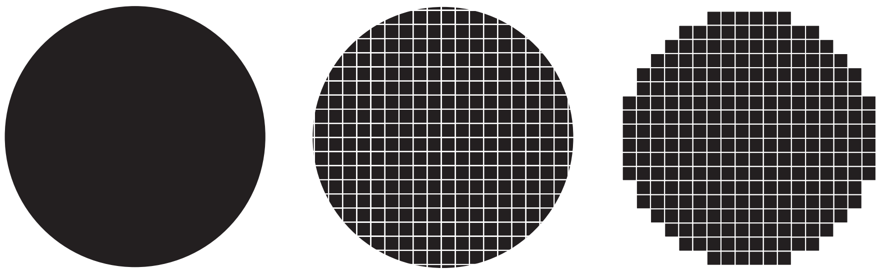

Figure 1.8: Quantizing data to fit into a computer. The circle on the left represents continuous data. The circle in the middle shows the grid upon which the data will fit once its values are made discrete. The circle on the right shows the final quantized result displayable on a computer. Likewise, audio must be quantized when stored digitally.

### 1.2.2 Classifying pitches from audio files

The *Fourier transform* is a mathematical algorithm that can be used to extract frequencies from a series of amplitudes. Most modern audio files are sampled at 44,100 $Hz$, which means that there are 44,100 data points for every second of audio. Let $sr$ denote the sampling rate of an audio file in $Hz$. For a given segment of audio consisting of $n$ data points, the Fourier transform returns $n$ values, where the magnitude of the $i$th value corresponds to the strength of the frequency $\frac{sr \cdot i}{n} Hz$. A graph of these values with time along the x-axis, frequency along the y-axis, and intensity represented by color is called a *spectrogram*. An example spectrogram of the Beatles song *Eleanor Rigby* is given in Figure 1.9.

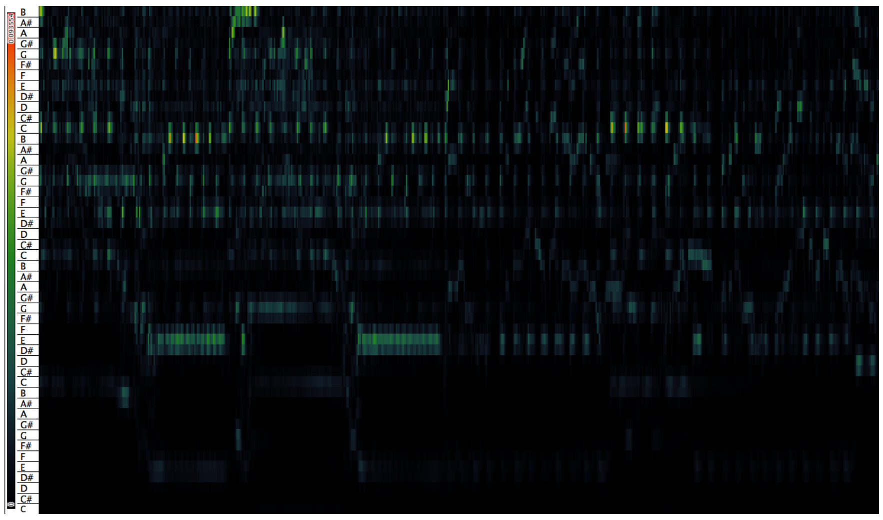

Figure 1.9: A spectrogram of an excerpt of The Beatles song *Eleanor Rigby*. The x-axis moves linearly with time, and the y-axis has been adjusted to a logarithmic scale to move linearly with a chromatic scale. The magnitude of each frequency value can be seen by the intensity of the color in the spectrogram.

The Fourier transform can be applied to an audio file using a *sliding window* in which the data points are analyzed in chunks. The window is of a set size to contain a certain number of data points and traverses the data linearly in equal, potentially overlapping steps. The size of the window is a balance in precision—the larger the window size the finer the frequency resolution; the smaller the window size the more closely note onsets and offsets can be detected. This can be illustrated with a diagram of spectrograms sampled at different window sizes in Figure 1.10.

Pitch classification of a segment of audio corresponds to finding peaks in this array of magnitude values. Often, spectrograms are condensed into an octave invariant representation called a *chromagram* in which

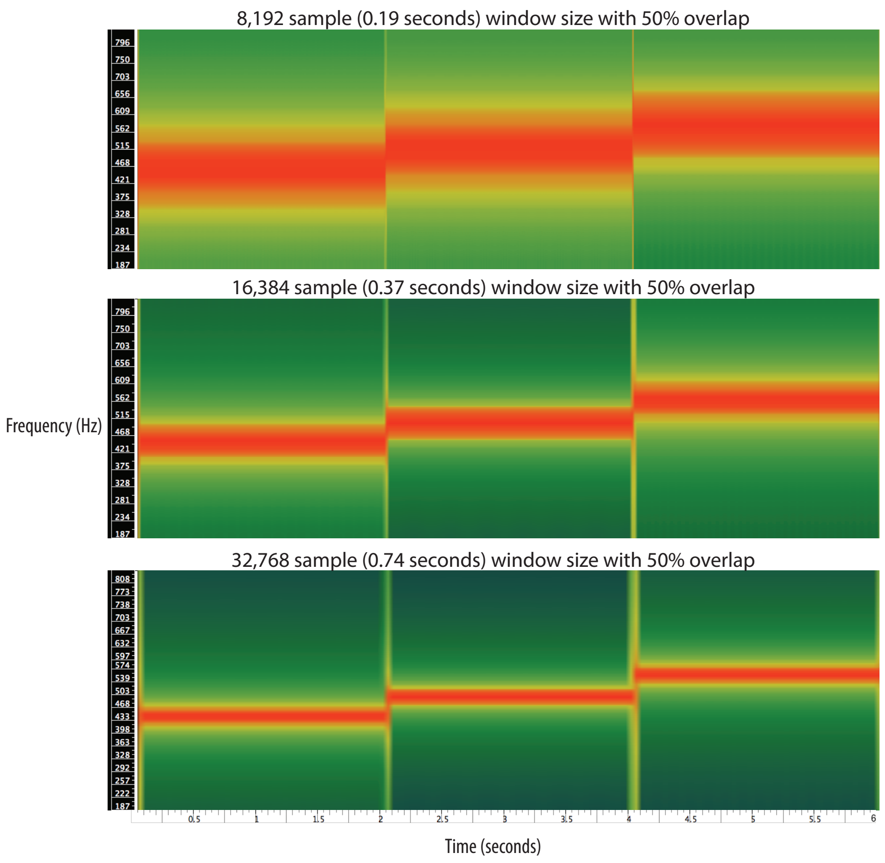

Figure 1.10: A spectrogram of Fourier transformations by window size on the same melodic line consisting of synthesized notes $A$, $B$, and $C\sharp$ sustained for two seconds each. As the window size increases, the width of each band depicting the notes being played narrows, corresponding to more attuned frequency resolution. Likewise, the vertical band representing error in identifying transitions between notes shrinks as window size decreases.

each note across all octaves is compounded into a single bin. A chromagram of the same excerpt of *Eleanor Rigby* can be found in Figure 1.11.

Automatic pitch identification of audio is inexact. The peak-finding approach only captures salient frequency values, which does not necessarily imply that the corresponding pitches are present. The complex, rich sound of instruments and voices have *overtones*, or frequencies that are multiples of a base frequency, presenting obscure samples. Noise and extraneous sound clutter recordings. The quantization of audio files and impreciseness of recording equipment prevent perfect data collection. Multiple melodic parts can make it hard to isolate regions or discern between instrumental lines. There are more complicated means of identifying pitches that take into account *timbral* aspects of instruments, color and overtones, but the

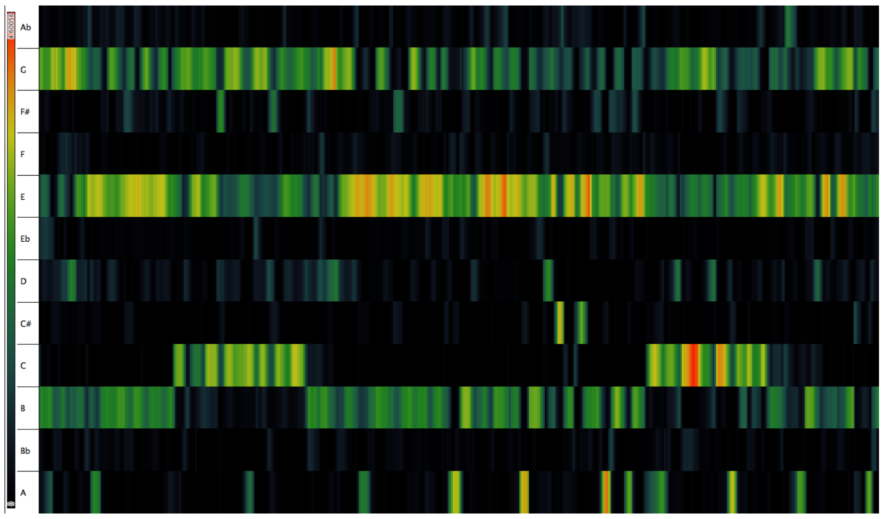

Figure 1.11: A chromagram of an excerpt of The Beatles song *Eleanor Rigby.* The x-axis moves linearly with time, and the y-axis represents pitch class. Notice the pitch classes $(\mathbf{B}, \mathbf{E}, \mathbf{G})$ and $(\mathbf{C}, \mathbf{E}, \mathbf{G})$ are the most prevalent vertical intensities at different time values. The chords *Emin* and *C* have these respective note representations and are the guiding harmonies in the song.

subjective nature of pitch interpretation means definitive truths are hard to establish, and the computational task of hearing as a human might broaches the field of artificial intelligence.

### 1.2.3 Classifying chords from audio files

Chord identification from audio files is a difficult task that would seem to compound the inexactness of pitch recognition and musical data collection into a more error-prone procedure; however, advanced chord recognition algorithms perform better than approaches that identify chords through individual notes. These more complicated techniques rely on harmonic templates, approximate transcription, and artificial intelligence.

Chord recognition algorithms commonly analyze *chroma features*, which represent the observed intensity of each pitch class in a segment of audio by compounding intensities of frequencies across different octaves into a single bin. Algorithms can use chroma features at different window positions to automatically study songs of known chord progressions and construct a model that learns the patterns of chroma features for each chord. This method describes *supervised machine learning*, in which a model is trained on a selection of ground-truth data and tested for accuracy on unseen ground-truth data. The underlying model in chord recognition is usually a *Hidden Markov model* (HMM) or *dynamic Bayesian network* (DBN), statistical graphs that depict probabilistic data and have hidden states to represent unknowns such as the chord being identified. The probabilistic relations described by the graph are updated as different chord data are observed, culminating in a trained model that can classify chords by finding the *Viterbi* paths, or most likely paths, through the graphs. The advantage of using machine learning is that the model can automatically take into account acoustical ways in which certain chords sound that are unexplained by just observing the strongest intensities within the chroma features. For instance, overtones of the notes within a chord can pollute the chroma feature vector in ways that the model can learn to understand [28]. Systems that use supervised machine learning with chroma features to classify chords are frequently employed [24, 25] and often extended substantively with MIR techniques that take into account other properties of the song, such as rhythmic beat information.

Matthias Mauch and Simon Dixon designed a chord recognition system in 2010 [28] that was state-of-the-art at the time of its introduction, having the highest chord identification as measured by *chord symbol recall*[2] in the 2010 MIREX Audio Chord Estimation contest[3]. Mauch and Dixon's approach contains a number of improvements on the traditional systems. Their algorithm preprocesses the input audio to remove inharmonious background noise. It then uses separate beat-synchronous chromagrams for both bass and treble features, differentiating analysis of lower and upper frequency registers with a window size synchronized to the rhythmic pulse of the song. Each chromagram is further divided into three bins per pitch class shifted in 1/3 of a semitone increments to infer the global tuning of the piece based on the strongest intensities present. The algorithm relies on approximate physical understandings of simultaneously played notes to determine the onset of each note and uses DBNs that have hidden states for the metric position of a chord, the song's key, and the bass notes being played. In later work, Mauch et al. observe higher levels of a song's structure such as repeated sections with similar chord information to improve overall chord recognition accuracy [30].

The research led by Maksim Khadkevich in [21] presents methods that currently have the best chord symbol recall in the 2014 MIREX Audio Chord Estimation contest[4]. Khadkevich uses *time-frequency reassignment* (TFR), a technique that "sharpens" spectrograms by remapping cells (spectral coordinates defined by frequency and time position) to reflect the regions of strongest support. This provides much better time-frequency resolution, rectifying the problem inherent in spectrograms as described in Subsection 1.2.2 and

[2]Chord symbol recall is simply the percentage $\frac{\text{duration of correctly annotated chords}}{\text{total duration of song}}$

[3]`http://nema.lis.illinois.edu/nema_out/mirex2010/results/ace/`

[4]`http://www.music-ir.org/mirex/wiki/2014:Audio_Chord_Estimation_Results`

exemplified in Figure 1.10. The algorithm then uses separate trained HMMs for each chord quality, presenting a robust system to classify chords from audio.

Chord classification brings multiple disciplines within computer science and music cognition together to make informed predictions on the chord sequences present in an audio file. There is inherent ambiguity in harmonic classification, even among experts, but existing chord identification algorithms perform well with regard to human-annotated ground-truth data. Chord classification techniques also require a specific alphabet of chord qualities—though this collection of all possible identifiable chord qualities is usually a modular component of a chord recognition algorithm, it presents a necessary design decision. Some use an expanded palette of chord symbols that includes extended chords in jazz such as *9ths*, *11ths*, and *13ths* [6]. Others consider only looking at whether the chord is *major* or *minor* without any extensions [34]. The alphabet of chord qualities labels presented in Figure 1.4 presents a good compromise between too much and too little information and is used in the chord identification systems presented in this paper, as described in chapter 3.

# Chapter 2

# Harmonic Metrics

In chapter 1 notions of theory, human subjectivity, and analytical technique were shown to govern classification of information from musical songs. The overview described how systems can be constructed from the ground up to classify, or extract, chords from a music audio file. This chapter details ways in which extracted chord labels and sequences of chords can be treated as computational objects for comparison, establishing the base tools that will build up to this paper's primary experimentation.

## 2.1 Chord Distance Metrics

### 2.1.1 Considerations

To begin to compare progressions of chords, it is essential to first have metrics to compare individual chords. Recall that a chord has the following components: *root*, *quality*, *bass*. Chord extraction algorithms also include a symbol to describe the absence of any chordal content in a segment of audio that can arise during silences or inharmonic sections of a song: *nochord*. There are multiple approaches to comparing two chords. Both chords can be represented as the set of their constituent pitch classes and set operations can be performed to formulate a distance; another approach may emphasize different structures such as the relationship between two chords' *root* notes and *bass* notes.

It can be helpful to outline some basic comparisons of chords and describe subjective expectations how the chord metric should perform to outline the utility of a distance function. These expectations are somewhat arbitrary but are founded on my subjective experiences with Western music and extracted chord results.

Assume a chord distance function $C_d$ takes two chords and returns a real number from 0, indicating no similarity, to 1, indicating perfect similarity:

- $C_d(C, Cm)$ should have a good similarity score since a common chord extraction error is to confuse the major and minor versions of a given chord based on the similarity of overtone content or ambiguity that is frequently used as a musical device. In terms of the pitch classes of $C$ major and $C$ minor, they are similar in that both have $C$ and $G$ but differ in that the former has $E$ and the latter $Eb$.

- $C_d(C, Am)$ should compare similarly as the chords are *relatively similar*, which means that the key signatures formed by taking their roots and using their qualities as modes describe the same set of pitch classes. The pitch classes in $A$ minor have two overlapping elements with those in $C$ major.

- $C_d(C, Cmaj7)$ should score well as $C$ major 7 is an extension of $C$ major, which means it has the same root and bass and contains a superset of the pitch classes in $C$.

- $C_d(C6, Am7)$ should compare nearly perfectly since both chords have the exact same set of pitch classes but differ in root and bass, which could be an error in chord extraction.

- $C_d(C7, C7/Bb)$ also can be expected to perform well as both chords describe identical sets of pitch classes and have the same root but differ in bass note.

#### 2.1.1.1 Notation

$P_c$ represents all the pitch classes in a given chord. $P_c(nochord)$ is the empty set, $\{\}$. *root* is a function that returns the pitch class of the root of the given chord. *quality* is a function that returns the qualitative label of a chord that can be used with other functions to extract intervals and other factors. *bass* is a function that returns the pitch class of the bass of the chord. *nochord* is a function that returns whether the given chord is **nochord** or not. $I$ is a function that takes a quality and returns a set of intervals in semitones modulus 12 from the root (excluding the root). $V$ is a function that takes a quality and returns the 5th interval in the chord. For chords with major, minor, and dominant qualities, regardless of extension, this is the semitone 7 steps above the root. For diminished chords and minor 7 flat 5 chords, this is the semitone 6 steps above the root. Finally, for augmented chords this is the semitone 8 steps above the root.

### 2.1.2 Simple equality test

This is the most basic implementation of a distance metric. Essentially, it tests whether two chords are equal, returning 1 if so and 0 otherwise. Equality can be tested in multiple ways. *Exact equality* first examines *nochord*, returning 1 if both chords are **nochord**; otherwise, it returns whether the root, quality, and bass of the two chords are identical. Exact equality can be expressed

$$C_d(c_1, c_2) = \begin{cases} 1 & \text{if } (nochord(c_1) \wedge nochord(c_2)) \vee \\ & \quad (root(c_1) = root(c_2) \wedge quality(c_1) = quality(c_2) \wedge bass(c_1) = bass(c_2)) \\ 0 & \text{otherwise} \end{cases}$$

where $\wedge$ is the boolean *and* operator and $\vee$ is the boolean *or* operator.

There are other ways to test equality. For instance, only root can be considered if one wishes to reduce a chord to its simplest element as follows:

$$C_d(c_1, c_2) = \begin{cases} 1 & \text{if } (nochord(c_1) \wedge nochord(c_2)) \vee root(c_1) = root(c_2) \\ 0 & \text{otherwise} \end{cases}$$

Another reasonable implementation is to consider the set of pitch classes of both chords and whether they are equal. This is expressed:

$$C_d(c_1, c_2) = \begin{cases} 1 & \text{if } P_c(c_1) = P_c(c_2) \\ 0 & \text{otherwise} \end{cases}$$

These simple equality metrics are useful distance functions for simple implementations and computations and were handy in assessing preliminary results before more detailed distance functions had been implemented.

### 2.1.3 Harte Distance Metric

This chord function is presented without a name in Christopher Harte's PhD thesis about extraction of harmony from audio [18], but I use the name "Harte distance metric" to refer to it. The method is deceptively simple but effective, consisting of taking the size of the intersection of both chords' pitch class sets and dividing it by the size of the union of their pitch class sets, as follows:

$$C_d(c_1, c_2) = \frac{|P_c(c_1) \cap P_c(c_2)|}{|P_c(c_1) \cup P_c(c_2)|}$$

Using the Harte distance metric, the score of comparing minor and major chords of the same root (e.g. $C_d(C, Cm)$) and a major chord with its relative minor (e.g. $C_d(C, Am)$) both have the same value of 0.5. A major chord compared with a major 7th of the same root (e.g. $C_d(C, Cmaj7)$) that contains a superset of the first chord's pitch classes has a score of 0.75. Comparing chords with identical pitch sets, even if root or bass notes vary (e.g. $C_d(C6, Am7)$ $C_d(C7, C7/Bb)$), results in a perfect score of 1.

This metric has the advantage that it is very versatile and fast, but potentially loses some information about a chord in disregarding distinctions of elements within the chord such as *root* and *bass*, instead focusing only on the pitch classes within.

### 2.1.4 Tonal Pitch Step

Tonal Pitch Step is a distance measure that is grounded in cognitive psychology, algebra, and tonal music theory, proposed by Fred Lerdahl in [26]. The measure is unique from the others in that it takes into account the key of a song, which can be recalled as:

$$Key \rightarrow Root\ Mode$$

$$Root \rightarrow NoteName$$

$$Mode \rightarrow \text{Major Minor}$$

Given all possible combinations of a pitch class describing *root* and *mode*, there are 24 distinct key signatures. It is important to note that the acceptable pitch classes within *relative* keys are identical, such as $C$ major and $A$ minor.

The tonal pitch step algorithm is calculated, with revisions courtesy of [10] to only include one key, as

$$C_d(c_1, c_2, k) = i(root(c_1), root(c_2)) + j(c_1, c_2, k)$$

where $i$ calculates the *circle-of-fifths* distance between two pitch classes. The circle-of-fifths (see Figure 2.1) distance describes how many $\pm 7$ semitone traversals are needed to reach one note from another. It can be calculated for any two pitches $p1$ and $p2$ as

$$i(p1, p2) = \min(((p1 - p2) \cdot 7) \mod 12, ((p2 - p1) \cdot 7) \mod 12)$$

where the difference between $p1$ and $p2$ is the number of semitones between them.

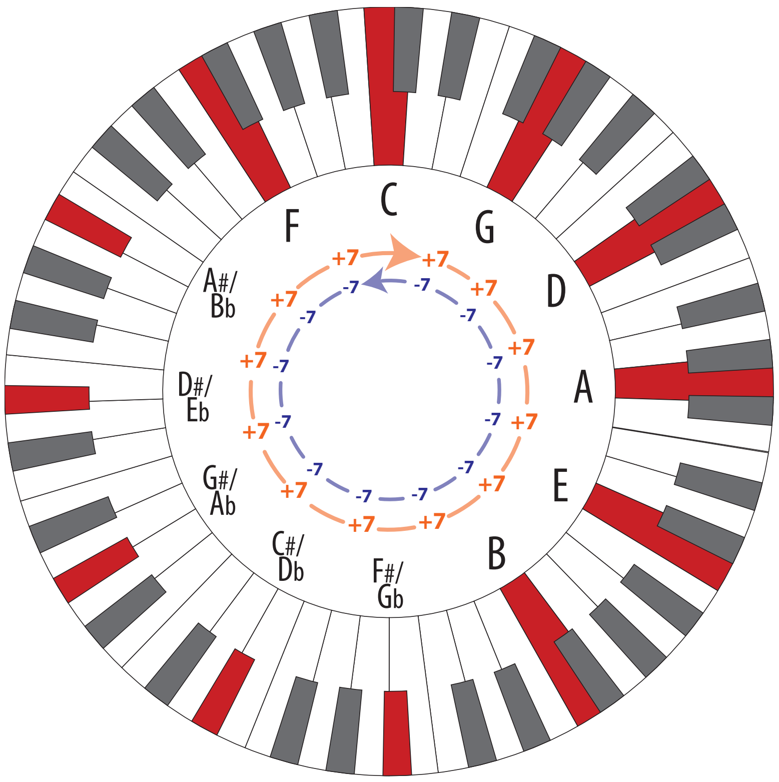

Figure 2.1: The circle-of-fifths. Keys highlighted in red are spaced 7 semitones apart, an interval referred to in music theory as a *fifth*. Each successive note clockwise corresponds to an interval of 7 semitones, and each step counterclockwise corresponds to a -7 semitone jump. The circle-of-fifths distance is used as a component in Lerdahl's Tonal Pitch Step metric.

The calculation of $j$ consists of deriving four sets of tonal space for each chord Lerdahl outlines in which each subsequent level is a superset of the previous level. Level 1, $L_1$, of a chord $c$ returns a set only containing the chord's root note:

$$L_1(c) = \{root(c)\}$$

Level 2, $L_2$, returns a set containing a given chord's root note and fifth interval:

$$L_2(c) = \{root(c), V(c)\}$$

Level 3, $L_3$, returns a set containing all the pitch classes in a given chord:

$$L_3(c) = P_c(c)$$

Finally, level 4, $L_4$, consists of all the pitch classes in the key signature given:

$$L_4(c, k) = P_c(k)$$

With each level, two sets, one from each chord being compared, are used and the cardinality of the *symmetric difference* ($\triangle$) is computed. The symmetric difference corresponds to the size of the set of unique elements between both sets being compared. For instance, $\{1,2,3\}\triangle\{2,3,4\} = \{1,4\}$. $j$ is calculated as follows:

$$j(c_1, c_2, k) = \frac{|L_1(c_1)\triangle L_1(c_2)| + |L_2(c_1)\triangle L_2(c_2)| + |L_3(c_1)\triangle L_3(c_2)| + |L_4(c_1, l)\triangle L_4(c_2, l)|}{2}$$

Tonal Pitch Space (TPS) returns a value from 0.0 to 13.0 (in the case of more advanced key signature modes the upper bound is 14.0) in which lower scores correspond to higher similarity between two chords. To make this measure comparable with the Harte distance metric, it is a matter of normalization and inversion to give TPS a range of 0 to 1 in which 1 indicates perfect similarity:

$$C_d(c_1, c_2, k) = \frac{13.0 - (i(root(c_1), root(c_2)) + j(c_1, c_2, k))}{13.0}$$

In the key of $C$ major, some example TPS results are calculated based on the chord comparisons of interest.

$$\begin{aligned}
C_d(C, Cm, C \text{ major}) &= 0.884 && (11.5 \div 13) \\
C_d(C, Am, C \text{ major}) &= 0.462 && (6.0 \div 13) \\
C_d(C, Cmaj7, C \text{ major}) &= 0.962 && (12.5 \div 13) \\
C_d(C6, Am7, C \text{ major}) &= 0.538 && (7.0 \div 13) \\
C_d(C7, C7/Bb, C \text{ major}) &= 1.0 && (13.0 \div 13)
\end{aligned}$$

Intriguingly, the comparison of $C6$ and $Am7$ has a significantly lower score of 0.538 than its perfect score under the Harte metric. The key information required in TPS penalizes $Am7$ in the context of $C$ major even though both $C6$ and $Am7$ describe the same set of pitch classes.

TPS does not take significantly longer to compute than the Harte metric and is grounded in more cognitive models of human chord perception, though this does not imply it is necessarily a better algorithm for chord comparisons.

#### 2.1.4.1 Key Finding Using Tonal Pitch Step

One pitfall of TPS is that it requires advance knowledge of the key signature over which chords are being compared. Given a sequence of chords $s$ this factor can be estimated by finding the key $k$ that maximizes the sum of the TPS distance of every chord in $s$ against the chord describing the key. Labeling the algorithm $K_e$, key finding can be expressed as

$$K_e(s) = \max_k \sum_c^s C_d(c, one(k), k)$$

where $k$ iterates through all 24 combinations of key signature root and mode, and the function $one$ returns the chord describing the given key with the root of $k$ and the quality corresponding to the key's mode.

In the paper *Comparing Harmonic Similarity Measures* [15], where this algorithm was proposed, the authors found that this formula produced many false positives, and accordingly modified the algorithm for use with the Western music they were testing to incorporate information about the $IV$, $V$, and $vi$ chords of the tested key (in major modes; in minor modes, this is expressed as the $iv$, $V$, and $VI$ chords). This additional information gave enough data about the salient harmonies in a key to have an accuracy rate of 88.8% over the corpus of music tested by the authors. Functions to derive these chords are based on a root and mode of a key signature. Given a function *chord* that constructs a chord with a given root and mode, these functions are as follows:

$$
\begin{aligned}
one(root, mode) &= chord(root, mode) \\
four(root, mode) &= chord((root + 5) \mod 12, mode) \\
five(root, mode) &= chord((root + 7) \mod 12, \text{major}) \\
six(root, mode) &= \begin{cases} chord((root + 9) \mod 12, \text{minor}) & \text{if } mode = \text{major} \\ chord((root + 8) \mod 12, \text{major}) & \text{if } mode = \text{minor} \end{cases}
\end{aligned}
$$

Let *rank* denote the rank function that returns the ranking of an element in a list or sequence (see Figure 3.5 for a more in-depth treatment of *ranking*). The rank corresponds to the number of elements less in value than that one, such that $rank(1, [2, 3, 1]) = 0$, $rank(2, [2, 3, 1]) = 1$, and $rank(3, [2, 3, 1]) = 2$. $r$ is a function that takes in a sequence, one of the key-to-chord functions ($\{one, four, five, six\}$), and the key being tested. It then returns the ranking of the key being tested out of all possible keys' summed TPS distances between every chord in $s$ against the key-to-chord function of the key, as follows:

$$r(s, c_f, k_0) = rank(k_0, \{k : \sum_{c}^{s} C_d(c, c_f(k), k)\})$$

The revised key finding algorithm takes a summation of $r$ calculations involving every key-to-chord function, with an emphasis on *one*, and incentivizes matching first and final chords with the key, returning the key that satisfies:

$$K_e(s) = \min_{k_0}\left(4 \cdot r(s, one, k_0) + r(s, four, k_0) + r(s, five, k_0) + r(s, six, k_0) + \begin{cases} -4 & \text{if the first chord of } s \text{ matches } k_0\text{, and} \\ -4 & \text{if the last chord of } s \text{ matches } k_0 \end{cases}\right)$$

In my implementation, a chord was determined *matching* in the last sum of this formula if its raw TPS score (0.0-13.0) was at most 2.0 when compared with $one(k_0)$.

## 2.2 Chord Progression Comparisons

### 2.2.1 Considerations

Computationally comparing chord progressions and approximating subjective human notions of similarity is a difficult task. Influential or unique chord progressions are frequently discussed in cultural contexts [35]. Looking for local similarities between chord progressions has proved effective [16] and seems to match human intuition. When a pair of songs are said to be similar it does not mean that both songs' progressions are similar globally, or throughout the entire song, but rather that there are identical or near-identical local sections of similarity. For instance, The Beatles song *Let It Be* has a progression of $I - V - vi - IV$ occurring at the beginning and frequently during the song. The chorus starts with a different progression but *Let It Be* is still classed with songs of the progression $I - V - vi - IV$ and is included on Wikipedia's "List of songs containing the I–V–vi–IV progression," which at the time of writing includes 189 songs [51]. Another famous progression, $vi - IV - I - V$ (which is a cyclic shift of the previous progression), has even been dubbed the "Sensitive Female Chord Progression" for its use in countless pop songs [19].

Chord progression identity is a salient factor in harmonically understanding a song, and preserving notions of local similarity is essential. The following sections detail algorithms that can be used to compare two chord progressions, their advantages and disadvantages, and preliminary experimental results, if any.

#### 2.2.1.1 Notation

The following notation will be used:

| | |
|---|---|
| $c1$ and $c2$ | The chord progressions of the two songs being compared, respectively |
| $c_i$ | The $i$th element of chord progression $c$ |
| $n1$ and $n2$ | The length of $c1$ and $c2$, respectively |
| $C_d$ | A chord distance function that takes two chords and returns a value in which higher scores indicate stronger similarity |
| $t_s$ | A transposition function that takes a chord as an argument and transposes it $s$ semitones |

### 2.2.2 Simple Global Comparison

This is perhaps the most basic implementation of chord progression comparison. For two songs with chord progressions of an identical length $n$, the algorithm iterates through one chord from each song simultaneously and computes $C_d$. The resulting score is:

$$\sum_{i=0}^{n} C_d(c1_i, c2_i)$$

Since the songs may be in different keys, the algorithm can iterate over all 12 transpositions of a song and pick out the maximal result. With the chord transposition function $t_s$, the algorithm can be written:

$$\max_{s=0}^{12} \sum_{i=0}^{n} C_d(c1_i, t_s(c2_i))$$

For songs of unequal length $n1$ and $n2$, respectively, the algorithm can be revised by computing the maximum resulting score at each position of the shorter song's chords shifted along the longer song's chords. Assuming the second song is the shorter song ($n2 < n1$), the revised resulting score can be computed:

$$\max_{s=0}^{12} \max_{i=0}^{n1-n2} \sum_{j=0}^{n2} C_d(c1_{i+j}, t_s(c2_j))$$

The advantages of the simple global comparison are that the algorithm is simple to implement and compute. Its computation costs are relatively low with a linear running time in the case of equal chord progression lengths. Unfortunately, in unequal song length comparisons, the running time is quadratic in terms of both songs' chordal lengths. These computation times are assuming a constant-time distance function.

The disadvantages of global comparison are that it neglects the power of local comparisons and does not correct well for errors in automated chord progression analysis. For instance, let $c1$ be the chord progression $(Am, Dm, E7, Am, ..., Am, Dm, E7, Am)$, where the four chords $(Am, Dm, E7, Am)$ repeat 100 times. Let $c2$ be the same chord progression but with an extra $E$ chord after the 50th iteration due to a flawed extraction. The global comparison score using $c1$ and $c2$ will be roughly cut in half due to that singular error and the issue with the ensuing alignment, with more errors resulting in potentially worse scores.

### 2.2.3 N-Gram Comparison

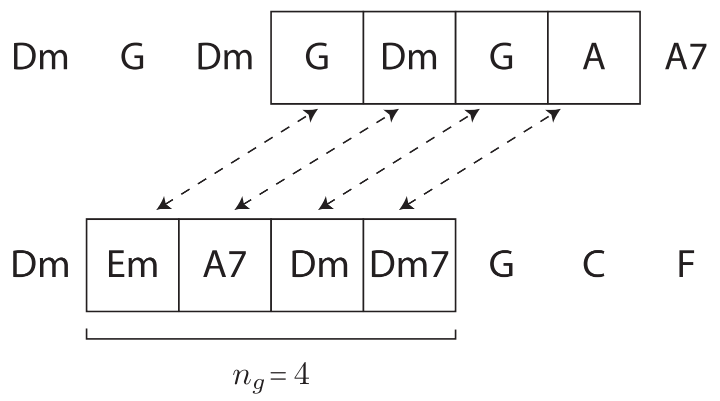

Figure 2.2: An example of sliding n-gram windows of length 4 traversing two songs. At the given position, both 4-grams would be compared by summing the chordal distance between each pair of chords represented by the dotted arrows.

To attempt to enhance the power of localized comparisons, *n-grams* can be used to measure distances between chord subsequences. An n-gram is just a sequence of a fixed length $n$, where 2-gram would refer to a sequence of length 2, 3-gram would refer to a sequence of length 3, and so on. Using two sliding windows fixed at length $n_g$, both songs can be traversed to calculate chordal distances between every n-gram in both songs and sum the distances. The chordal distance between two n-grams can be calculated by summing the chordal distance of each vertical pair of chords within the n-gram (see Figure 2.2), essentially the global comparison algorithm of equal length applied within a small window. This can be expressed:

$$\sum_{i=0}^{n1-n_g} \sum_{j=0}^{n2-n_g} \left( \sum_{k=0}^{n_g} C_d(c1_{i+k}, c2_{j+k}) \right)$$

To take into account songs in different keys, transpositions can easily be factored in to find the key with the maximal score:

$$\max_{s=0}^{12} \sum_{i=0}^{n1-n_g} \sum_{j=0}^{n2-n_g} \left( \sum_{k=0}^{n_g} C_d(c1_{i+k}, t_s(c2_{j+k})) \right)$$

An advantage of using n-grams is that all local regions of similarity are included, even if it's the last $n_g$ chords of one song and the first $n_g$ of another. A disadvantage is the running time of the algorithm, which is quadratic and in practice proved to be slower than other measures. Another disadvantage is that the window size, $n_g$, must be decided ahead of time and thus limits the flexibility of the algorithm. Finally, there is still not a good means of compensating for small errors, like scattered occurrences of erroneous chords. N-gram count metrics are considered for profiling in [35] and for melodic retrieval in [50].

I implemented a simple Python program that calculates this algorithm using 4-grams and the Harte chord distance metric (see Section 2.1.3) to show pairwise comparisons between extracted chord progressions of the songs *Let It Be* by The Beatles, *When I Come Around* by Greenday, and a live rendition of *Let It Be*. All these songs have the same base chord progression, but I expected *Let It Be* and its live rendition to have the maximal similarity. For the maximal key transposition between each comparison of the songs, I collected every pair of 4-grams between each comparison, placed the distances in bins, and plotted each comparison as 3 histograms. The maximum distance between any two 4-grams is 4.0, and I chose a bin size of 0.2. On the resulting graphic, the x-axis corresponds to the 4-gram distance bins, and the y-axis corresponds to the frequency of occurrence of all the n-grams that were placed in the bins in the pairwise comparison between the two songs. I also list the average 4-gram distance and the frequency of 4-gram's with distances greater than or equal to 3.0 to observe the rate of occurrence of more similarly matched 4-grams. The transposition in semitones used to obtain the maximal score displayed is listed above each histogram. I manually checked the key of each song and found that both *Let It Be* renditions to be in the key of $C$ major and *When I Come Around* to be in $F\#$ major, corresponding accurately to the transpositions shown. The resulting graphic is in Figure 2.3.

### 2.2.4 Minimum Edit Distance

Minimum edit distance, or the *Levenshtein distance*, is a distance metric that computes the minimum number of operations needed to transform one sequence into the other using *deletion*, *insertion*, and *substitution* [27]. Deletion entails removing an item from the sequence being transformed, insertion adding one item to the sequence, and substitution replacing an item from the sequence with another item. Each operation has an associated cost, and the function finds the minimal cost to transform one sequence into the other.

The technique is most commonly applied to *string matching* in which two strings of text are compared. For instance, assuming all operations are of equal cost, the minimum number of transformations needed to transform `MORDENT`, a rapid alteration of notes, to `MODESTO`, a modest musical mood, is 3 as exemplified in Figure 2.4.

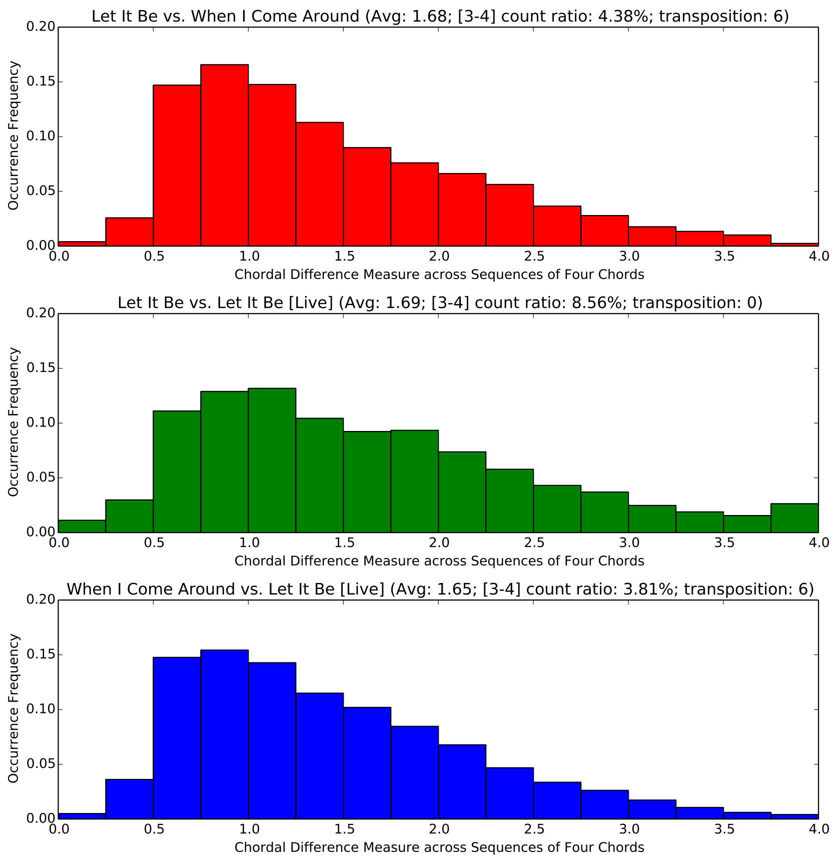

Figure 2.3: A histogram plot of 4-gram comparisons between 3 songs using the Harte distance metric. Higher x-values correspond to higher n-gram similarity scores, with a 4.0 corresponding to a perfect match. The two renditions of *Let It Be* (middle graph) have 8.56% of their values with scores of at least 3.0, the highest of any of the comparisons.

`MORDENT`

Apply *deletion* on `R`:

`MODENT`

Apply *substitution* to change `N` to `S`:

`MODEST`

Apply *insertion* to add an `O` at the end:

`MODESTO`

Figure 2.4: A minimum edit distance transformation from `MORDENT` to `MODESTO` with a cost of 3.

This alignment can be represented as follows:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| M | O | R | D | E | N | T | * |
| \| | \| | Del | \| | \| | Sub | \| | Ins |
| M | O | * | D | E | S | T | O |

Though minimum edit distance is frequently run on plaintext strings, there are no limitations to its alphabet. It can be easily abstracted to the alphabet of musical chords and is done so as a harmonic chord progression similarity measure in [20].

The advantages of minimum edit distance are that it can compensate for small errors at the global level of comparison. For instance, if there are two sequences of chords that are identical except for one minor error of transcription in the middle of the first chord progression, minimum edit distance will only penalize the error with one deletion operation, a cost of 1, rather than an exponential decrease in score as is the case with simple global comparison (see Section 2.2.2). Additionally, minimum edit distance is a *dynamic programming* algorithm, which means it solves a seemingly complex problem by breaking it down into subproblems, and its overall runtime is quadratic.

The primary disadvantage of minimum edit distance is that it can only be used to find global alignments between songs and does not harness the power of local alignments.

### 2.2.5 Smith-Waterman

The Smith-Waterman algorithm [44] combines the power of minimum edit distance with extensible substitution and gap costs, serving to find the region of optimal local similarity between two sequences rather than the global. This means that the algorithm can locate, similar to the n-grams approach, similar subsequences at all positions of both sequences and optimize a similarity measure; unlike n-grams, Smith-Waterman also optimizes all possible lengths of subsequences. Like minimum edit distance, the algorithm can be solved in quadratic time as it breaks down the complex task into computationally tractable subproblems. Though often used in bioinformatics and molecular applications with DNA or Protein alphabets, there is no reason the alphabet used cannot be abstracted to chord symbols.

The algorithm runs with the following functions, $S$ and $W$. $S$ is the substitution function, also referred to as a *cost matrix*, and defines the cost of transforming one symbol to the other. $W$ refers to the gap function and assigns a cost to a gap of integer length, where a gap is essentially a combination of insertion or deletion operators. It is common to define a $gap_{open}$ constant and a $gap_{extension}$ constant such that

$$W(i) = \begin{cases} (-gap_{open} - gap_{extension}) \cdot (i-1) & \text{if } i >= 1 \\ -gap_{open} & \text{if } i = 0 \end{cases}$$

Given two sequences $a$ and $b$ of length $m$ an $n$, a matrix $H$ of size $m \times n$ is constructed in the following manner by first populating the first row and column with zeros ($H(x,y)$ is the matrix element retrieval notation where $x$ corresponds to column and $y$ corresponds to row):

$$H(i,0) = 0, 0 \le i \le m$$

$$H(0,j) = 0, 0 \le j \le n$$

Subsequently, the matrix is traversed from the top-left across rows to the bottom-right, building off of previous elements with a recurrence relation as follows:

$$H(i,j) = \max \begin{cases} 0 \\ H(i-1, j-1) + S(a_i, b_j) \\ \max_{k \ge 1} \{H(i-k, j) + W(k)\} \\ \max_{l \ge 1} \{H(i, j-l) + W(l))\} \end{cases}$$

The maximal element of the resulting matrix, $\max H$, defines the Smith-Waterman score ($SW$), which corresponds to the maximum score within any two localized regions between two sequences based on the cost of transformation and gap penalties. This localized region can be thought of as two sliding windows, one for each song, similar to the approach in the n-grams chord progression comparisons; however, in this approach, each window allows transformative operators and can be of arbitrary size. The combination of window sizes and positions in each sequence that maximizes the resulting score is chosen.

#### 2.2.5.1 Example

To retrieve positional information from a completed $H$ matrix, it is helpful to look at an example of the Smith-Waterman algorithm in action. Let the following chord progressions be represented as sequences $a$ and $b$:

$$a = [F, C, Dm, G, F, C, Dm, C, F]$$
$$b = [C, F, C, G, F, C, G, Dm, C]$$

This example uses an application of the simple equality chord distance measure

$$C_d(c_1, c_2) = \begin{cases} 4 & \text{if } root(c_1) = root(c_2) \vee (nochord(c_1) \wedge nochord(c_2) \\ -3 & \text{otherwise} \end{cases}$$

and a simplified gap cost of:

$$W(i) = -i$$

The matrix is constructed with an empty first row and column (represented with a —) and then completed according to the recurrence relation:

$$\left(\begin{array}{ccccccccccc}
 & - & Cmaj & Fmaj & Cmaj & Gmaj & Fmaj & Cmaj & Gmaj & Dmin & Cmaj \\
- & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
Fmaj & 0 & 0 & 4 & 0 & 0 & 4 & 0 & 0 & 0 & 0 \\
Cmaj & 0 & 4 & 0 & 8 & 4 & 0 & 8 & 4 & 0 & 4 \\
Dmin & 0 & 0 & 1 & 4 & 5 & 1 & 4 & 5 & 8 & 4 \\
Gmaj & 0 & 0 & 0 & 0 & 8 & 4 & 0 & 8 & 4 & 5 \\
Fmaj & 0 & 0 & 4 & 0 & 4 & 12 & 8 & 4 & 5 & 1 \\
Cmaj & 0 & 4 & 0 & 8 & 4 & 8 & 16 & 12 & 8 & 9 \\
Dmin & 0 & 0 & 1 & 4 & 5 & 4 & 12 & 13 & 16 & 12 \\
Cmaj & 0 & 4 & 0 & 5 & 1 & 2 & 8 & 9 & 12 & 20 \\
Fmaj & 0 & 0 & 8 & 4 & 2 & 5 & 4 & 5 & 8 & 16
\end{array}\right)$$

The numbers colored blue show the elements that were used in the path to the maximal score, 20, colored red. The path can be backtracked using arrows:

$$\left(\begin{array}{ccccccccccc}
 & - & Cmaj & Fmaj & Cmaj & Gmaj & Fmaj & Cmaj & Gmaj & Dmin & Cmaj \\
- & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
Fmaj & 0 & 0 & \nwarrow & 0 & 0 & 4 & 0 & 0 & 0 & 0 \\
Cmaj & 0 & 4 & 0 & \nwarrow & 4 & 0 & 8 & 4 & 0 & 4 \\
Dmin & 0 & 0 & 1 & \uparrow & 5 & 1 & 4 & 5 & 8 & 4 \\
Gmaj & 0 & 0 & 0 & 0 & \nwarrow & 4 & 0 & 8 & 4 & 5 \\
Fmaj & 0 & 0 & 4 & 0 & 4 & \nwarrow & 8 & 4 & 5 & 1 \\
Cmaj & 0 & 4 & 0 & 8 & 4 & 8 & \nwarrow & \leftarrow & 8 & 9 \\
Dmin & 0 & 0 & 1 & 4 & 5 & 4 & 12 & 13 & \nwarrow & 12 \\
Cmaj & 0 & 4 & 0 & 5 & 1 & 2 & 8 & 9 & 12 & \nwarrow \\
Fmaj & 0 & 0 & 8 & 4 & 2 & 5 & 4 & 5 & 8 & 16
\end{array}\right)$$

Enumerating the path from the red arrow results in the following sequence of arrows

$$[\nwarrow, \nwarrow, \leftarrow, \nwarrow, \nwarrow, \nwarrow, \uparrow, \nwarrow, \nwarrow]$$

which can be applied to a bijection

$$operationFromArrow(arrow) = \begin{cases} substitution & \text{if } arrow = \nwarrow \\ deletion & \text{if } arrow = \leftarrow \\ insertion & \text{if } arrow = \uparrow \end{cases}$$

and reversed to produce the sequence:

$[substitution, substitution, insertion, substitution, substition, substitution, deletion, substitution, substitution]$

To show the optimal local alignment between both sequences, the appropriate subsections of both sequences need to be extracted. From $a$ this corresponds to the chords whose rows feature blue and red text

$$[F, C, Dm, G, F, C, Dm, C]$$

and from $b$ this corresponds to the chords corresponding to the columns that feature blue and red text

$$[F, C, G, F, C, G, Dm, C]$$

Like the example in minimum edit distance, both sequences are aligned with one another. *Substitution* operators are represented with vertical lines between corresponding elements. Insertion operators indicate a gap in the first sequence (represented with *) and a chord in the second sequence. Finally, *deletion* operators correspond to a gap in the second sequence and a chord in the first sequence. $a$ and $b$ can then be stitched together, or *aligned*, as follows:

$$\begin{array}{ccccccccc} F & C & Dm & G & F & C & * & Dm & C \\ | & | & \text{Ins} & | & | & | & \text{Del} & | & | \\ F & C & * & G & F & C & G & Dm & C \end{array}$$

#### 2.2.5.2 Use as a Chord Progression Similarity Measure

Smith-Waterman is useful in the context of comparing chord progressions as it has mechanisms to deal well with inexact data, using different gap costs and chord distance substitution functions that compensate for small errors. Both the Harte distance metric and TPS can be easily used as a substitution function. For a given distance function $C_d$, gap costs $gap_{open}$ and $gap_{extension}$, the Smith-Waterman algorithm is:

$$\max_k^{12} SW(c1, t_k(c2), C_d, W_i)$$

where $C_d$ corresponds to the $S$ substitution cost function in the Smith-Waterman algorithm and $W_i$ the gap penalty function:

$$W(i) = \begin{cases} -gap_{open} - gap_{extension} \cdot (i-1) & \text{if } i >= 1 \\ 0 & \text{if } i = 0 \end{cases}$$

Essentially, the score that is returned is the maximum of all twelve transpositions of one sequence relative to the other. It is important to have an expected value below 0 for the chord distance metric for use with Smith-Waterman so that regions of similarity are localized. This can be accomplished by scaling and subtracting an existing metric by a certain factor (see Section 4.2.3).

Because of all its advantages and research that supports its efficacy [16], the Smith-Waterman algorithm will be used to compare chord progressions in this paper as discussed in chapter 3. There are downsides to the Smith-Waterman algorithm. In its current form it can only be used to extract one optimal local alignment, and the score returned reflects only that local alignment, whereas n-grams comparison does well with multiple like regions of local similarity. There are adjustments to the algorithm to return multiple good alignments, but they prove computationally expensive. Allali et al. [2] describe a process for constructing a 3-dimensional Smith-Waterman algorithm that can account for *modulations*, or transpositions to a new key signature mid-song. These adaptations leave room for future experimentation. This paper focuses on only returning one good alignment using the optimized Smith-Waterman implementation discussed in Section 4.2.

#### 2.2.5.3 Normalization

A difficulty with the Smith-Waterman algorithm is that of comparing scores from different Smith-Waterman results. *Normalization* measures refer to attempts to make raw Smith-Waterman scores, which have a positive correlation with increased sequence length, invariant of sequence length through formulaic and statistical means. For notational convenience, $SW(c1, c2)$ will refer to a Smith-Waterman chord progression comparison score between two chord progressions without bothering with notating the chord distance and gap penalty function.

Statistical techniques comprise of collecting many samples of $SW$ scores of the sequences being tested in different shuffled permutations and deriving mean and standard-deviation factors [45]. I wrote a program to collect this kind of data using two chord progressions and constructed a line plot of the result (see Figure 2.5) in which $SW$ score is along the x-axis and number of occurrences out of 100,000 samples is on the y-axis. As can be seen, based on the chord distance function used (in this case simple equality), odd and even $SW$ score values have drastically different empirical probabilities of occurring. I did not use this technique in my experimentation, but it could well be explored in future work.

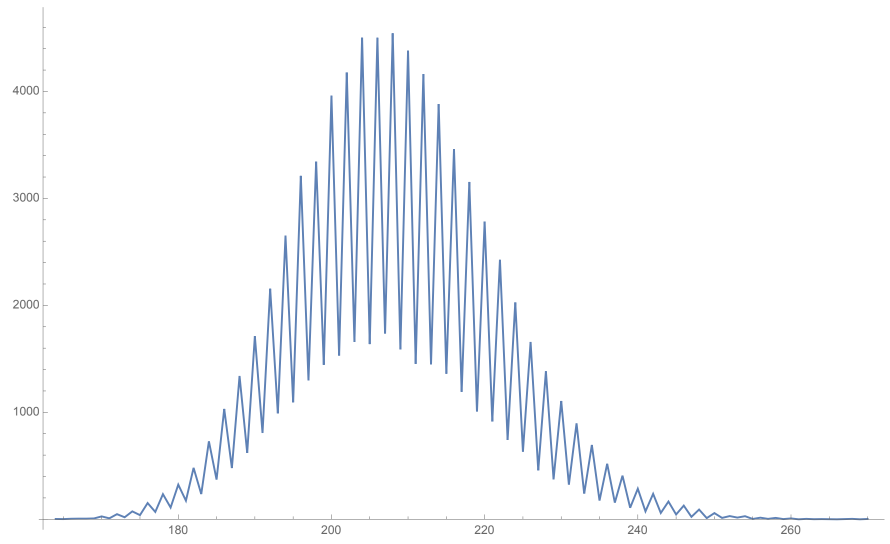

Figure 2.5: A line plot representing the score distribution for a Monte Carlo simulation of Smith-Waterman scores for random permutations of the input data over 100,000 trials. The x-axis corresponds to Smith-Waterman score, and the y-axis is number of occurrences.

Another measure to normalize the Smith-Waterman score is

$$SW_{norm}(c1, c2) = \frac{SW(c1, c2)}{\max[SW(c1, c1), SW(c2, c2)]}$$

which involves running the Smith-Waterman algorithm over chord progressions against themselves. This returns the score of identity in the distance metric multiplied by the length of the sequence being tested, which has a direct linear correlation with sequence length. Essentially, this is linearly equivalent to a normalization by dividing by the maximal length of the sequences being tested and returns values from 0 to 1.

The only normalization measures considered in this paper are *raw score* and this measure I call $SW_{norm}$.

# Chapter 3

# Methodology

## 3.1 Overview of Experimental Design

This paper tests how similarly various harmonic music informatics retrieval tasks perform using extracted chord data versus human produced data. A collection of musical songs is acquired for which verified human ground-truth annotations exist. An established chord identification algorithm is run with default settings to produce a set of extracted chord annotations. Two different primary harmonic comparison tasks are performed in isolated experiments on the ground-truth and extracted chord annotations separately—full pairwise chordal connections between all songs and query by n-gram. These harmonic comparison tasks are performed using the Smith-Waterman sequence alignment algorithm (2.2.5) and the Harte or TPS chord distance metric (2.1). In each harmonic comparison task, lists of ground-truth and extracted comparison results are produced. These lists are ranked and correlated using established ranking metrics, producing a correlational harmonic metric between the two lists of results. These procedures will be outlined in greater depth in the following sections.

In order to maximize the correlation between these different annotated datasets that describe the same musical content, I use global optimization techniques to search for optimal variables for the Smith-Waterman algorithm and chord distance metrics, repeating and assessing many iterations of the same harmonic comparison tasks with changing parameters. As the search space is too large to perform an exhaustive evaluation, the returned set of parameters and corresponding correlational metric present a good approximation of the optimum configuration. The goal of this experiment is to show that computationally extracted chord annotations perform similarly to ground-truth annotations in harmonic comparison tasks that use traditional chord sequence alignment algorithms and distance metrics with well-chosen parameters. This process is outlined in a flowchart in Figure 3.1.

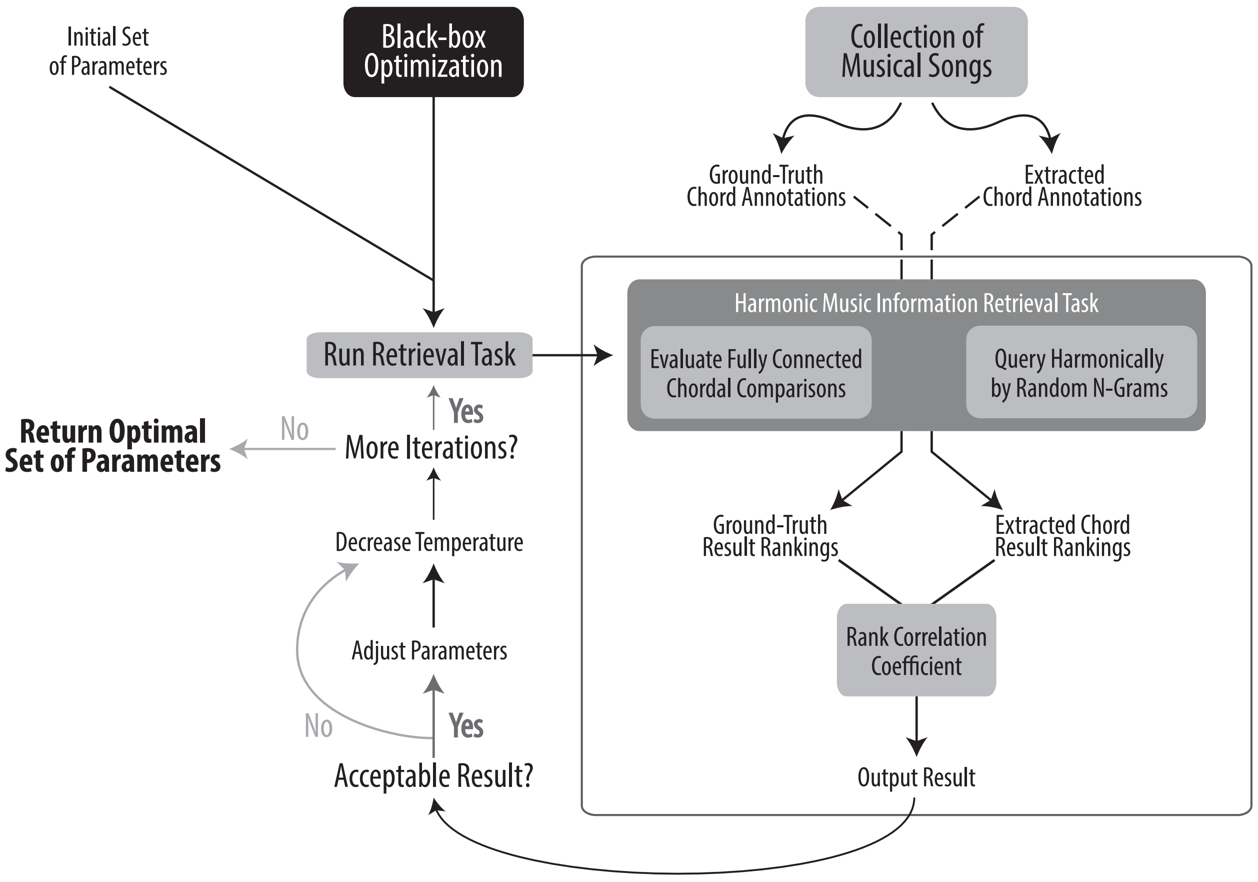
Initial Set of Parameters
Black-box Optimization
Collection of Musical Songs
Ground-Truth Chord Annotations
Extracted Chord Annotations
Harmonic Music Information Retrieval Task
Evaluate Fully Connected Chordal Comparisons
Query Harmonically by Random N-Grams
Run Retrieval Task
Yes
No
Return Optimal Set of Parameters
More Iterations?
Ground-Truth Result Rankings
Extracted Chord Result Rankings
Decrease Temperature
Rank Correlation Coefficient
Adjust Parameters
No
Yes
Acceptable Result?
Output Result

Figure 3.1: Flowchart of experimental design. This experiment requires a collection of songs with corresponding ground-truth and computationally extracted chord annotations. These different chord datasets describing the same collection of songs are fed into a harmonic retrieval task in isolated experiments, each producing a different result list. These result lists are ranked and correlated to return a correlational harmonic metric. Global optimization techniques search for maximum correlational harmonic metrics by running many iterations of the retrieval task with changing parameters based on how well the correlational result is relative to previous iterations. The returned set of parameters represent an approximate optimal configuration for minimizing algorithmic differences between human and computationally extracted chord inputs.

Each of these tasks and parameters will be outlined in more depth in the following sections, along with details of the datasets and their collection process. This chapter primarily serves to overview the paper's design choices and considerations, setting up the background knowledge for and mechanics of the experiments covered in chapter 5. Many of the experimental choices made are arbitrary but are guided by intuition. I will discuss rationales and limitations, along with possible extensions.

## 3.2 Chord Extraction

### 3.2.1 Chordino

*Chordino*[1] is an open-source chord extraction software program written by Matthias Mauch based off his winning 2009 and 2010 MIREX chord estimation algorithm submissions [7, 28]. Chordino achieves an 80% chord symbol recall and is still considered state-of-the-art [32]. The working mechanics of the algorithm are detailed in Subsection 1.2.3. Though Khadkevich's algorithm [21] currently has the highest chord symbol recall in the 2014 MIREX audio chord estimation task, there is not publicly released source code for his work. Chordino is intended for the public and has been made available in the form of a plugin for the *VAMP*[2] plugin system architecture which enables different audio analysis programs to access provided external analytical methods. Additionally, the primary dataset of extracted human chord annotations used by my experiments (3.3) was compiled in 2011. Unlike Khadkevich's chord identification algorithm released in 2014, there is no possibility that Chordino could have been influenced by or tested against this dataset, maintaining a purity of separation between data and system.

Chordino is the only chord extraction algorithm considered in this paper and is used with default settings. In these settings, chords are returned within a finite set of qualities consistent with those detailed in Figure 1.4. Further experimentation would benefit from testing different chord extraction algorithms at a range of settings and variant alphabets of chord qualities. As is, my paper leaves room for expansion and demonstrates the efficacy of Chordino settings used in their initial configuration, solely optimizing for parameters in the Smith-Waterman algorithm and chord distance metrics.

[1]http://isophonics.net/nnls-chroma
[2]http://www.vamp-plugins.org/

## 3.3 Datasets

### 3.3.1 McGill Billboard Annotations

The *McGill Billboard* annotations collected in [6] and freely available online[3] are a state-of-the-art human-annotated chord dataset. The dataset is comprised of over 1,000 songs sampled from different decades from the 1950s to the early 1990s across different Billboard charts from the United States "Hot 100" [4]. The researchers hired music experts and professional jazz musicians to annotate the songs randomly sampled from the Billboard charts, financially incentivizing the participants and creating an online system that managed the task and rewarded songs that were more difficult to transcribe. Each song was annotated twice to maintain a standard of accuracy. The resulting dataset is the most comprehensive current ground-truth set of chord annotations and is used in recent MIREX chord annotation competitions. Importantly, the dataset was released in 2011, which postdates the Chordino chord extraction algorithm used in experimentation (see Section 3.2) and thus obviates the possibility of training bias (that the algorithm was trained to produce good results with these songs).

In my implementation I describe how I obtained the corresponding audio files from which to extract chord annotations (see Section 4.3.3). 529 songs' audio files were successfully collected corresponding to ground-truth annotations. This led to the creation of the primary datasets used in this experiment—the ground-truth McGill dataset and the extracted McGill dataset.

#### 3.3.1.1 Chord Alphabet Used

The ground-truth McGill dataset uses a detailed chord alphabet with 99 classes of distinct chord qualities. Though the website provides simplified chord annotations, the qualities used in these annotations are not consistent with the qualities used in Chordino by default. To maintain the greatest precision, a simplification process can be outlined to convert McGill chord annotations to Chordino chord qualities to have a consistent chord quality alphabet used throughout the experimentation. For each chord $c$ and given a function $chordFromParts(root, quality, bass)$ that constructs a chord from its components, the algorithm proceeds as

$$\mathit{Simplify}(c) = \overset{\text{Chordino qualities}}{\min_{q}} \quad Harte(chordFromParts(root(c), q, bass(c)), c)$$

[3] http://ddmal.music.mcgill.ca/billboard

[4] http://www.billboard.com/charts/hot-100

where `Chordino qualities` refers to the set of chord qualities used in Chordino (see Figure 1.4), essentially finding the chord with the same root and bass notes that has the minimal Harte distance with $c$.

The simplification process may seem to lower the precision afforded by manually annotated chord labels and deliberate human consideration. Apart from consistency, the motivation of the simplification is to not have too detailed a semantic depiction of chords (see the last paragraph of 1.2.3), though it is noted this could be a potential limitation of this experiment. Future experimentation could test leaving these chord labels as is and using richer chordal depictions.

## 3.4 Chord Progression Comparison

To compare chord progressions, I use the Smith-Waterman local sequence alignment algorithm (2.2.5). I test two distance functions with the Smith-Waterman algorithm: Harte distance (2.1.3) and Tonal Pitch Space (2.1.4)—these will be referred to algorithmically as $Harte$ and $TPS$.

### 3.4.1 Smith-Waterman

The Smith-Waterman algorithm takes several parameters: $gap_{open}$, $gap_{ext}$, and $C_d$ (2.2.5.2). Due to implementation details of the Smith-Waterman algorithm (4.2), the $gap_{open}$ and $gap_{ext}$ parameters are restricted to certain ranges and integer values. While the implementation allows values up to 255, I chose 128 arbitrarily as a maximal range. The result score can also optionally be normalized (2.2.5.3)—the parameter $norm$ will be used and can be either $raw$ (no normalization) or $SW_{norm}$. These parameters are charted in Figure 3.2.

| Parameter | Range |
|---|---:|
| $gap_{open}$ | $[0 - 128]$ (inclusive) |
| $gap_{ext}$ | $[0 - 128]$ (inslusive) |
| $C_d$ | $\{Harte, TPS\}$ |
| $norm$ | $\{raw, SW_{norm}\}$ |

Figure 3.2: Smith-Waterman Parameters

### 3.4.2 Chord Metrics

The Harte and TPS chord metrics can be normalized to fall within a range from 0 to 1, where 1 indicates perfect similarity and 0 indicates no similarity. Let this quantity be denoted $cnorm(C_d)$. Subsection 4.2.3 introduces measures $m_x$ and $m_s$ used to round a chordal distance to an integer for use with the Smith-Waterman implementation. $m_x$ denotes a multiplier constant and $m_s$ a subtraction constant, such that the rounded chord distance $C_{rd} = round(cnorm(C_d) \cdot m_x) - m_s$. This introduces two additional parameters which I have restricted to the following ranges (see Figure 3.3):

| Parameter | Range |
|---|---|
| $m_x$ | $[1 - 30]$ (inclusive) |
| $m_s$ | $[0 - 30]$ (inslusive) |

Figure 3.3: Rounded Chord Parameters

As the expected value of the chord distance function $C_d$ has to be negative to isolate localized alignments, these values are further restricted.

#### 3.4.2.1 Testing Negative Expected Value for Rounded Chord Distance

To ensure the expected value of the rounded chord distance function $C_{rd}$ is negative, I iterate through all possible chord-to-chord comparisons and verify that the average ($\frac{\text{sum}}{\text{length}}$) is negative. For simplicity, this can be tested with just the sum:

$$\sum_{c_1}^{\text{all chords}} \sum_{c_2}^{\text{all chords}} C_{rd}(c_1, c_2)$$

This is the approach I use; however, sampling chords that actually occur in the songs being tested may provide a more meaningful notion of experiment-dependent expected value. Sampling for expected value should be explored in future work.

## 3.5 Harmonic Comparison Tasks

This section describes two high-level tasks that form the substance of the experiments. Inputted with parameters and a database of chord annotations for a collection of songs, these algorithms perform chord progression comparisons using the harmonic metrics previously outlined. The result is a collection of harmonic comparison scores that can be enumerated in an ordered fashion as a sequence.

These tasks are used in a more high-level optimization process. Each task is run separately with ground-truth chord annotations and computationally extracted chord annotations. The two sequences corresponding to the resulting comparison scores are ranked and assigned a correlation score with traditional statistical methods. This final ranked score, a *correlational* metric of harmonic similarity, is cycled into a global optimization process. This section is not concerned with these higher level processes or the distinction between the ground-truth and computationally extracted chord annotations. Each task can be viewed singularly in terms of taking a dataset of chord annotations and associated harmonic parameters and returning a sequence of results.

### 3.5.1 Fully Connected Pairwise Harmonic Comparison

This first retrieval task is the most simple—given parameters and a dataset of chord annotations, this method returns every pairwise combination of songs' chord progression comparison scores. The parameters inputted to this task are the Smith-Waterman gap parameters (see Figure 3.2) and the rounded chord parameters (see Figure 3.3). The algorithm starts by iterating through each pairwise connection between any two chord progressions in the dataset, proceeding in an ordered and well-defined fashion such that results are consistent. The algorithm takes each pair of songs' chord progressions and calculates the Smith-Waterman chord progression similarity score using the inputted parameters, returning the results to a list in the same ordering used for iteration.

This task is called *fully connected* pairwise harmonic comparison in reference to a fully connected graph in which each node is connected to every other node. If the songs are represented as nodes, each edge corresponds to a chord progression comparison. A visual depiction of a fully connected graph and the iteration procedure is illustrated in Figure 3.4.

Given a Smith-Waterman algorithm that takes a parameter set $p$ and two songs $c_1$ and $c_2$ as argument and a dataset of songs $D$ of length $n$, the fully connected pairwise harmonic comparisons between two songs in $D$ can be enumerated without redundancies:

$$\{\{SW(p, D_i, D_j) : i + 1 \leq j \leq n\} : 0 \leq i \leq n - 1\}$$

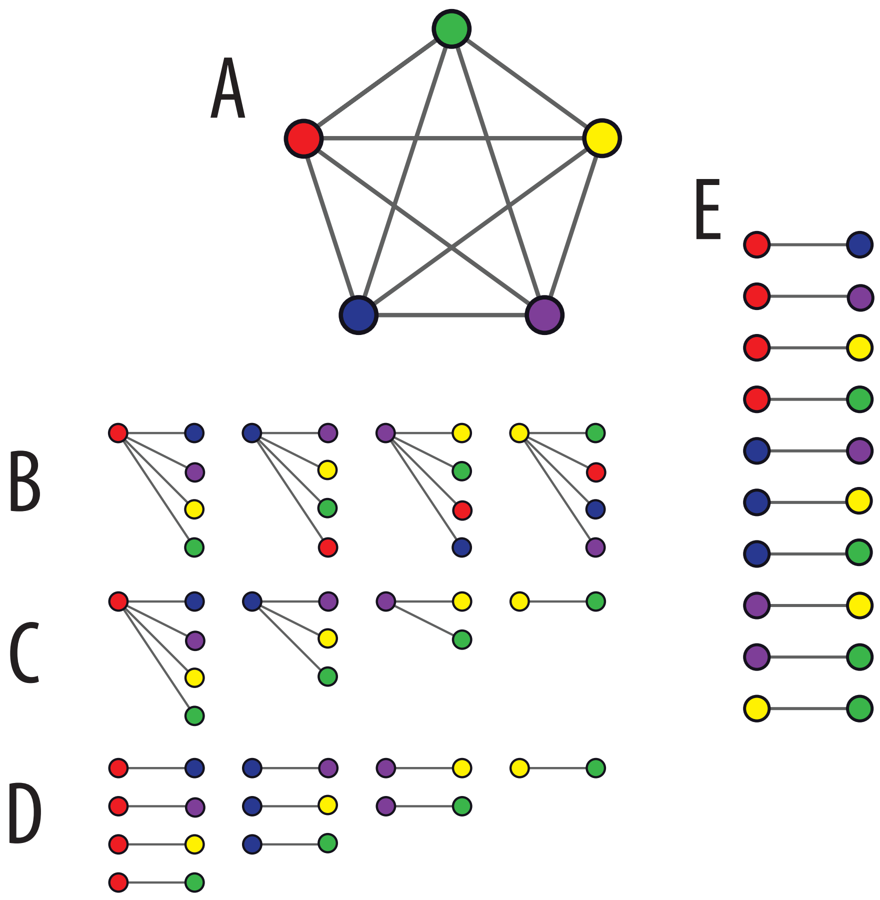

Figure 3.4: Enumerating edges in a fully connected graph. (A) is a fully connected graph with 5 colored nodes. Each node represents a song, and each edge represents the pairwise harmonic comparison between two songs. These edges are normally assigned a weight, or number, that describes the chord progression similarity score between the two connecting songs; these weights are not visually depicted. To enumerate all pairwise comparisons, one node is selected arbitrarily and all edges from that node are listed. Starting with the red node, for example, each edge and connecting node can be iterated counter-clockwise. Then, the same procedure can be repeated on the blue node, one step counter-clockwise from the red node, and so on until the graph is exhausted (B). It is not necessary to include redundant edges. For instance, the edge connecting the red and blue nodes and the edge connecting the blue and red nodes are identical. Redundancies can be eliminated by only listing each edge once and omitting repeat occurrences (C). The representation can now be expanded (D) and stacked (E) to produce an ordered sequence of comparisons that represent all connections in the fully connected graph (A).

Or in pseudocode:

```
function HarmonicComparisonSequence(D,n,p):
  result = []
  for (i = 0, i < n - 1, i++)
    for (j = i + 1, j < n, j++)
        result += [SW(p, D[i], D[j])]
  return result
```

### 3.5.2 Query by N-gram

This retrieval task involves searching a database of songs' chord progressions with a chord sequence query. The query sequence is compared with every song in the database and the maximal chord progression similarity score is returned. The query sequence is padded in length by repeating itself such that the length is at least that of the longest song. I chose this repetition of the query sequence to imitate the repetitive structure of musical songs and emphasize the cyclic nature of chord progression perception, though it is an arbitrary choice. Further experimentation could involve testing the effects of not including padding or using another means of expanding the query sequences.

As a singular query and its results provide limited data about the efficacy of a system, I decided to fabricate 100 query sequences, randomly generated initially but consistently used across experiments. I divided the 100 query sequences into 25 query sequences of lengths 4, 8, 16, and 32. For each query sequence, this method collects Smith-Waterman scores for each comparison of the query sequence with every song in the database, returning a 2-dimensional array of 100 sequences of Smith-Waterman score sequences. In pseudocode, my method can be expressed:

```
function NgramQuerySequences(D, n, ngrams, p):
  padNgrams(ngrams) // pad lengths of n-grams to at least length of longest song
  result = []
  for (i = 0, i < 100, i++)
    subresult = []
    for (j = 0, j < n, j++)
      subresult += [SW(p, ngrams[i], D[j])]
    result += [subresult]
  return result
```

## 3.6 Ranking and Correlating Results

### 3.6.1 Ranking of a sequence

Recall the ranking of a sequence is a mapping of every element of the sequence to its position in the sequence. For instance, $ranking([5, 2, 3]) = [3, 1, 2]$, since 5 is the 3rd element of the list in order, 2 is the 1st element, and 3 is the 2nd element. In the case of ties, half numbers are used at the midpoint of the ties, such that $ranking([1, 2, 3, 3, 4]) = [1, 2, 3.5, 3.5, 5]$ and $ranking([1, 2, 3, 3, 3, 4]) = [1, 2, 4, 4, 4, 6]$. The *ranking* method is explained in greater depth with an example in Figure 3.5.

Ranking can be applied to a sequence of chord progression similarity scores. For instance, consider the raw Smith-Waterman result data showing the fully connected chordal comparisons for the songs *When I Come Around*, by Greenday, and *Let It Be*, by the Beatles (studio and live version):

```
"Let It Be"            "When I Come Around"    102
"Let It Be"            "Let It Be [Live]"      108
"When I Come Around"   "Let It Be [Live]"      85
```

After the sequence is ranked, the output should resemble the following:

```
"Let It Be"            "When I Come Around"    2
"Let It Be"            "Let It Be [Live]"      3
"When I Come Around"   "Let It Be [Live]"      1
```

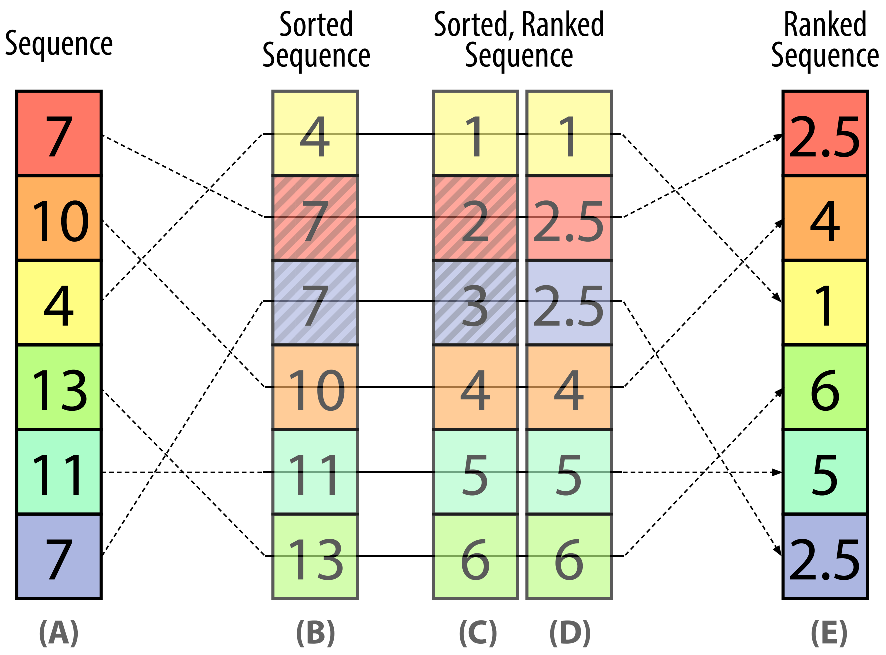

Figure 3.5: The method of ranking a sequence. First, all the elements of the sequence are sorted ($B$). Then, each element in the sorted sequence is assigned a rank starting at 1 and incrementing with each subsequent element ($C$). In the case of tied sorted elements, the ranks assigned are the average of the collective ranks of the sorted elements ($D$). This is demonstrated with the two 7's in the sorted column ($B$), assigned a rank of 2 and 3, respectively, which averages to two 2.5's ($D$). Lastly, all the sorted, ranked elements are rearranged back to their original positions, obtaining the ranking for the original sequence ($E$).

### 3.6.2 Metrics for evaluating the similarity between two rankings

To compare two identically sized ranked sequences $s_1$ and $s_2$ of length $n$, two metrics are used: $\rho$ and $\tau$. Both metrics return values from -1 to 1, with 1 indicating a perfect positive correlation, -1 a perfect negative correlation, and 0 no correlation. Two sequences are *monotonically increasing* if they have a positive correlation, and *monotonically decreasing* if they have a negative correlation.

#### 3.6.2.1 Spearman's Rho

*Spearman's rho* [13] ($\rho$) is the primary rank correlation metric I use for optimization. $\rho$ is inversely proportional to the square of the difference between two ranking sets and is also known as *Spearman's footrule.* $\rho$ is simple and can be evaluated in linear time as

$$\rho(s_1, s_2) = 1 - \frac{6\sum_i^n (s_{1_i} - s_{2_i})^2}{n(n^2-1)}$$

#### 3.6.2.2 Kendall's Tau

*Kendall's tau* ($\tau$) takes into account the number of inversions needed to correctly "swap" the two ranking sets such that they are the same. $\tau$ can be calculated as

$$\tau(s_1, s_2) = \frac{2\left((\text{number of identically ranked pairs}) - (\text{number of non-identically ranked pairs})\right)}{n(n-1)}$$

## 3.6.3 Use with Harmonic Retrieval Tasks

#### 3.6.3.1 Example

Now that metrics have been established to calculate the correlation between two sequences of data, these correlational metrics can be applied to chord progression similarity scores. Given the fully connected song comparisons used in 3.6.1, here is an example showing how *correlational harmonic metrics* (see 3.6.3.2) are established with different chord progression datasets describing the same songs:

```
=========Ground-Truth Chord Annotations=========          ------------Extracted Chord Features------------

"Let It Be"           "When I Come Around"   102          "Let It Be"           "When I Come Around"   106

"Let It Be"           "Let It Be [Live]"     108          "Let It Be"           "Let It Be [Live]"     105

"When I Come Around"  "Let It Be [Live]"     85           "When I Come Around"  "Let It Be [Live]"     92
```

After both sequences are ranked, the output is as follows:

```
=========Ground-Truth Chord Annotations=========          ------------Extracted Chord Features------------

"Let It Be"           "When I Come Around"   2            "Let It Be"           "When I Come Around"   3

"Let It Be"           "Let It Be [Live]"     3            "Let It Be"           "Let It Be [Live]"     2

"When I Come Around"  "Let It Be [Live]"     1            "When I Come Around"  "Let It Be [Live]"     1
```

Now that there are two sequences of rankings, rank metrics can be used to compare relative algorithm output similarity between inputted ground-truth and extracted chord annotations. Let $s_1$ be the ground-truth rankings $[2, 3, 1]$ and $s_2$ be the extracted chord rankings $[3, 2, 1]$. $\rho$ is calculated as follows:

$$\begin{aligned}\rho(s1, s2) &= 1 - \frac{6\sum_i^n (s_{1_i} - s_{2_i})^2}{n(n^2-1)} \\ &= 1 - \frac{6((2-3)^2 + (3-2)^2 + (1-1)^2)}{3(3^2-1)} \\ &= 1 - \frac{6(1+1+0)}{24} = 1 - \frac{12}{24} = \mathbf{0.5}\end{aligned}$$

The $\rho$ score between the two sets of rankings is thus 0.5, indicating a mild positive correlation. As the next section explains, I call this value a correlational harmonic metric. As the length of the input sequences increases, smaller positive correlation results become more statistically significant.

#### 3.6.3.2 Correlational Harmonic Metrics

As shown in the boxed region of the flowchart (Figure 3.1), ranking is a tool that is applied in this paper to the results of harmonic retrieval tasks. Ground-truth and extracted chord annotations corresponding to a collection of musical songs can be evaluated through one of the tasks and their result sequences can be ranked and then compared through one of the ranking metrics. In the case of *fully connected pairwise comparisons*, the resulting scores of both the ground-truth and extracted chord annotations are ranked separately, and then $\rho$ is used to return a similarity score. I call this resulting score a *correlational harmonic metric* as it reflects how similarly ground-truth and extracted chord annotations perform through a harmonic retrieval task.

In the *query by n-gram* task, each of the 100 query sequences across both the ground-truth and extracted datasets have a chord progression comparison results sequence. Each of these results sequences is individually ranked, then for each query sequence, $\rho$ is calculated using the ground-truth chord progression comparison rankings and the extracted chord progression comparison rankings. The returned correlational harmonic metric is the average of these 100 $\rho$ values, effectively calculating the average correlational harmonic metric for each query sequence across ground-truth and extracted chord datasets.

## 3.7 Global Optimization with Simulated Annealing

This section presents the final process of the flowchart (Figure 3.1)—*black-box optimization.* Optimization refers to the task of trying to find parameters to a problem that maximize or minimize the output. *Black-box* means that the problem, or function, being considered is not described in mathematical or algebraic terms. In my use of global optimization, the problem is one of the harmonic retrieval tasks, a function that takes a set of parameters $p$ and compares ground-truth and extracted chord progression comparisons, ultimately returning a correlational harmonic metric indicating how well that set of parameters produces well-correlated output results. The goal of optimization is to select good parameters such that the output of the algorithm correlates well across ground-truth and extracted chord datasets representing an identical collection of songs.

While mathematical intuitions may help select parameters, the experimental terrain is unknown and the behavior of the harmonic tasks are unexplained by simple models. The technique presented here, *simulated annealing*, is an established and simple optimization algorithm that makes no assumptions about the behavior of the function and is able to find good approximations of global optimums for the parameters.

### 3.7.1 Overview

Simulated annealing is an optimization technique that references heat treatment in metallurgy in which a material is warmed up and then cooled, hardening in the process [8, 22]. The computer optimization algorithm tries to minimize a function $f(s_t)$ in $i_t$ iterations, where $f$ returns a number and $s_t$ is a set of state variables. A function, $move(s_t)$, is applied with each iteration of the algorithm. $move$ performs some change to the inputted set of state variables $s_t$, returning a new slightly changed state and leaving the original value intact. $f$ is then recalculated with the new state to see if the move was fruitful based on its delta with the old state. The acceptable delta of values is called temperature, or $T$, and exponentially decreases with each iteration. If the $move$ results in a state with an $f$ that differs by more than $T$ from the previous iteration, that move is rejected and the state is left unchanged. This essentially gives the algorithm more exploratory freedom in the initial stages as the temperature $T$ is higher. The exponential decrease of the temperature $T$ results in a "cooling off" in which more and more moves are rejected. At the defined end of the algorithm, after $i_t$ iterations, the resulting state should be a good approximation of a minimum of $f$. The optimal $s_t$ that returns the minimum $f$ is returned, though this is not necessarily the final state. The algorithm can be outlined in pseudocode as follows:

```
function f(s_t) // energy function
  return ...

function move(s_t) // move function
  return ...
```

```
function simulatedAnnealing(init_state, i_t, T_start, T_end)
  s_t = init_state
  i = 0
  T = T_start

  T_factor = -log(T_start / self.T_end) // where log is the natural logarithm
  previous_f = null // keeps track of each previous f value
  min_f = null // stores the minimum f value encountered
  min_s_f = null // stores the associated state

  new_s_t = s_t
  while i < i_t
    // calculate next f
    new_f = f(new_s_t)
    // set a new minimum if the next f value is the lowest encountered
    if (min_f = null or new_f < min_f)
      min_f = new_f
      min_s_f = new_s_t
    // if the next f value is within the appropriate temperature range, update the state
    if (prev_f = null or abs(new_f - prev_f) <= T) // where abs is the absolute value
      s_t = new_s_t
      previous_f = new_f

    // update values at end of every iteration
    i = i + 1
    // set T to an exponential interpolation between the starting and ending temperatures
    T = T_start * exp(T_factor * i / i_t) // where exp is the natural exponential function
    // calculate next state
    new_s_t = move(s_t)
```

```
// return the minimum state and f values encountered
return min_s_f, min_f
```

### 3.7.2 Simulated Annealing for Harmonic Retrieval Experiments

To use simulated annealing to optimize different harmonic comparison tasks, the state $s_t$ can represent the different variables being used $\{norm, C_d, gap_{open}, gap_{ext}, m_x, m_s\}$. The *move* function in this case is intended to represent a transition to a nearby state—as each variable is an integer, the jump should be discrete. My implementation takes a random step following a normal distribution for each variable in the state, rounding the result to the nearest integer and ensuring the value falls within the bounds of the variable. The standard deviation of this random step for each variable is chosen to be a certain fraction of that specified variable's range, a constant denoted by the friction variable $f_r$. In the experiments run, the friction chosen was $\frac{1}{3}$. Variables that take two qualitative labels rather than numbers—*norm* and $C_d$—can be treated as integer variables taking a range of $[0-1]$ inclusive. With an $f_r$ of $\frac{1}{3}$, this represents approximately a 7% chance of a toggle of state from 0 to 1, or vice-versa; in practice, binary jumps occurred around 9% of the time in a single variable and 18% of the time in one or more of the variables *norm* and $C_d$ considered simultaneously. If a move results in a $m_x/m_s$ combination such that the expected value of the rounded chord distance function (3.4.2.1) is positive, the $m_x$ and $m_s$ components are randomized again from their last values following the same normal distribution jump process. This process is repeated until a negative expected value is obtained such that the Smith-Waterman algorithm can run as intended, isolating localized chord comparison results.

In experiments, I ran 1,000 iterations of simulated annealing as described above with a temperature $T$ that started at 1 and decreased exponentially to 0.005 at the final iteration. In the *fully connected comparison* (FCC) task, the optimization algorithm searched for maximum parameter sets to maximize $\rho$, the harmonic correlational metric. In *query by n-gram* (QBN), the optimization algorithm searched for maximum parameter sets to maximize the average $\rho$ across all 100 query sequences. It is important to note in QBN the queries were randomly selected only once at the start of the simulated annealing algorithm and consistently used throughout subsequent iterations.

# Chapter 4

# Implementation

This chapter discusses in brief the manner in which I implemented my code. The full project is over 6,000 lines of C, Python, HTML, and Javascript code. This chapter will only overview notable features of the code.

## 4.1 Using the chord alphabet

### 4.1.1 Integer representation of chords

The Smith-Waterman algorithm is typically used in bioinformatic applications in which the alphabet is restricted to DNA or protein characters. To use an alphabet that contains all the chord symbols, a bijective function can be established between every type of chord and a unique 16 bit integer. Recall a chord can be described with the following grammar:

$$\begin{aligned}
\mathit{Chord} &\rightarrow \mathit{Root}\ \mathit{Harmony}\ \mathit{Bass} \mid \textbf{NoChord} \\
\mathit{Root} &\rightarrow \mathit{PitchClass} \\
\mathit{Bass} &\rightarrow \mathit{PitchClass} \\
\mathit{PitchClass} &\rightarrow \textbf{A} \mid \textbf{A\#/Bb} \mid \textbf{B} \mid \textbf{C} \mid \textbf{C\#/Db} \mid \textbf{D} \mid \textbf{D\#/Eb} \mid \textbf{E} \mid \textbf{F} \mid \textbf{F\#/Gb} \mid \textbf{G} \mid \textbf{G\#/Ab} \\
\mathit{Harmony} &\rightarrow \textbf{maj} \mid \textbf{6} \mid \textbf{maj7} \mid \textbf{m} \mid \textbf{m6} \mid \textbf{m7} \mid \textbf{7} \mid \textbf{aug} \mid \textbf{dim} \mid \textbf{dim7} \mid \textbf{m7b5} \mid \textbf{UnknownHarmony}
\end{aligned}$$

Notice that $|Root| = |Bass| = |PitchClass| = 12$ and $|Harmony| = 12$. A bijective function $p$ between $PitchClass$ and an integer from 0 through 11 can be established, along with a bijective function $h$ between $Harmony$.

| $PitchClass$ | $p(PitchClass)$ |
|---|---|
| **A** | 0 |
| **A#/Bb** | 1 |
| **B** | 2 |
| **C** | 3 |
| **C#/Db** | 4 |
| **D** | 5 |
| **D#/Eb** | 6 |
| **E** | 7 |
| **F** | 8 |
| **F#/Gb** | 9 |
| **G** | 10 |
| **G#/Ab** | 11 |

| $Harmony$ | $h(Harmony)$ |
|---|---|
| **maj** | 0 |
| **6** | 1 |
| **maj7** | 2 |
| **m** | 3 |
| **m6** | 4 |
| **m7** | 5 |
| **7** | 6 |
| **aug** | 7 |
| **dim** | 8 |
| **dim7** | 9 |
| **m7b5** | 10 |
| **UnknownHarmony** | 11 |

In base 12, a chord that is not **NoChord** can be represented as an integer in which the digits are positioned as follows:

|  |  |  |
|---|---|---|
| $p(Bass)$ (0-11) | $p(Root)$ (0-11) | $h(Harmony)$ (0-11) |

This can be calculated as $(12 \cdot 12) \cdot p(Bass) + 12 \cdot p(Root) + h(Harmony)$. To include **NoChord**, the base 12 representation of a chord can be shifted by 1 and 0 can be reserved for **NoChord**, thus a bijective function $ChordToInt$ to map any chord $c$ to an integer can be calculated as follows:

$$ChordToInt(c) = \begin{cases} 0 & \text{if } c = \textbf{NoChord} \\ 144 \cdot p(Bass) + 12 \cdot p(Root) + h(Harmony) + 1 & \text{otherwise} \end{cases}$$

Extracting features from an integer $i$ representing a chord is then a simple task that can be represented in pseudocode as follows:

```
function ExtractFeatures(i)
  if (i = 0)
    return NoChord
  else
    Bass = (i - 1) / 144
    Root = ((i - 1) / 12) mod 12
    Harmony = (i - 1) mod 12
    return (Bass, Root, Harmony)
```

### 4.1.2 Bitwise representation of harmony

Chord quality can be represented as a 12-bit integer in which each bit corresponds to whether a certain interval is included. This compact form provides a means for quick computation and allows chord set operations to be expressed with bitwise operators.

Let $i$ be a 12-bit integer representing chord quality in which each bit corresponds to whether a certain interval is included in the chord or not. Since leading 0's are excluded, the representation can start with 1 in the first binary position (right-to-left) to represent the root of the chord and each subsequent binary position $i, 1 \leq i \leq 12$ can represent whether the interval $i$ is included (where 0 is the root note) (see Figure 1.4). All harmonies used in the program and associated intervals can be seen in Figure 4.1. Let $h_b$ be the function that extracts the binary mask from a given harmony.

| Chord Quality | Shorthand | Binary Representation | Base 10 Representation |
|---|---|---|---|
| Major | | 10010001 | 145 |
| Major 6th | 6 | 10001001 | 137 |
| Major 7th | maj7 | 1001001 | 73 |
| Minor | m | 100010001 | 273 |
| Minor 6th | m6 | 100010010001 | 2193 |
| Minor 7th | m7 | 10010001001 | 1161 |
| Dominant 7th | 7 | 10010010001 | 1169 |
| Augmented | aug | 1001001001 | 585 |
| Diminished | dim | 10001001001 | 1097 |
| Diminished 7th | dim7 | 1010010001 | 657 |
| Half-diminished 7th | m7b5 | 1010001001 | 649 |

Figure 4.1: Binary Representation of Chords

### 4.1.3 Bitwise chord operations

Basic chord operations can then be constructed. For instance, to get $P_c(c)$ as a 12-bit integer in which each bit $i$ from right-to-left corresponds to whether $p(i) \in P_c(c)$ one need only construct a bit cycling algorithm

```
function CycleBits(value, shift)
  return ((value << shift) | (value >> (12 - shift))) & b111111111111
```

where `b111111111111` refers to the binary bitmask of all 1's for 12 places, $<<$ and $>>$ are the bit shift left and right operators, and `|` and `&` are the bit operators *and* and *or*, respectively. $P_c(c)$ can then be calculated:

$$P_c(c) = CycleBits(h_b(quality(c)), p(root(c)))|p(bass(c))$$

Cleverness with representing chords and harmonies as integers allows effective methods to be constructed with simple bitwise operations and bitmasks.

## 4.2 Smith-Waterman SIMD Implementation

### 4.2.1 Initial implementations

To get my ideas off the ground, I initially implemented the Smith-Waterman algorithm for use with chords in Python, using high-level data types rather than integers to represent chords and calculating chord distances on the fly rather than pre-populating a distance matrix.

This implementation was effective for rapid prototyping, developing a deeper understanding of the algorithm, and being able to quickly add features like matrix output harnessing the power of Python's external libraries. Unfortunately, the code was extremely slow, taking in the ballpark of 4 seconds to compare two songs with around 150 chords each using adaptations of the simple equality chord distance metric. A limitation in the current research of chord sequence alignment systems (CSAS) is the slow runtime.

I attempted to improve the runtime by reimplementing the algorithm in its entirety in C, using bit representations of chords and precomputing distance matrices, achieving a speed increase of about 130x from the Python implementation. This speed compares with another paper that details implementation of a chord progression distance comparison based off of Smith-Waterman that had a runtime of 2-9 days to compare 5000 songs [15].

### 4.2.2 Adapting an external C implementation

To attempt to increase the speed of the Smith-Waterman algorithm even more, to leverage maximal algorithmic performance, I adapted a Smith-Waterman C implementation that uses Single-Instruction, Multiple-Data (SIMD) parallel computing instructions for use with the chord alphabet [52][1]. SIMD instructions, which almost all modern computer architectures have, operate on vectors of small data in parallel by representing the vectors as single integers. For instance, two 128-bit integers containing 16 8-bit integers can be operated on simultaneously to achieve tasks like adding all 16 of the 8-bit integers within each 128-bit integer in a single computer instruction.

The SIMD Smith-Waterman algorithm uses advances in Smith-Waterman calculations [9] in combination with SIMD instructions that can massively parallelize the algorithm at no additional computational cost. A problem I encountered was that this particularly SIMD implementation was only attuned for 8-bit integer alphabets preset to accommodate the few characters used in DNA sequences and protein sequences. I modified the source code to support the 16-bit representation I constructed of chords detailed in Section 4.1.1 while preserving the runtime of the algorithm, facilitating the use of the SIMD Smith-Waterman algorithm to chord progressions. To my knowledge, this is the fastest implementation of chord sequence alignment that currently exists. The runtime of this adapted algorithm compared with the initial Python implementation has a speed-up of over 113,000x. To compare my performance results with the researchers in [15] who achieved 2 days as their fastest fully connected harmonic comparison for a 5,000 song dataset, I constructed a dataset of 5,001 popular cover songs from Second Hand Songs[2] (see 4.3.5). My fully connected harmonic evaluation involved 149,970,000 iterations of the Smith-Waterman algorithm (12,497,500 iterations for each of the 12 transpositions) and took 18 minutes and 49 seconds, a speed-up factor of 154x. While this result is by no means official, the speed of this algorithm provides a greater leverage to run large-scale data experiments.

[1]https://github.com/mengyao/Complete-Striped-Smith-Waterman-Library
[2]http://secondhandsongs.com/

### 4.2.3 Rounding Chord Distance Metrics

Chord distance values, in the SIMD implementation of Smith-Waterman, are represented as 8-bit integers; this is incompatible with the floating point data the chord distance metrics proposed in Section 2.1. A workable solution is to take the normalized chord distance score $norm(C_d)$ and round it to the nearest integer after multiplying it

$$C_{rd} = round(C_d \cdot m_x)$$

for some multiplication factor $m_x$. To produce chord distance scores with an expected value below 0, a subtraction factor $m_s$ can also be introduced:

$$C_{rd} = (round(C_d) \cdot m_x) - m_s$$

## 4.3 Dataset Collection

*Scraping* refers to the task of writing scripts to automatically extract files and download them. What follows is a brief overview of how I collected, or scraped, the datasets I use in my primary evaluation and other minor veins of experimentation.

### 4.3.1 File conversion

To convert collected files—which come in a variety of audio formats such as *mp3*, *aac*, and *ogg*, and video formats such as *mp4*—to *wav* files necessitated using the external library *ffmpeg*[3]. ffmpeg contains code to convert a wide range of audiovisual file types to other file types. To feed chord data into Chordino via a command line script requires the use of *wav* files, so all input audio files were converted to *wav* for the purpose of chord extraction. I wrote a wrapper to convert files and analyze their chordal content with Chordino in Python.

[3]https://www.ffmpeg.org/

### 4.3.2 YouTube Extraction

This method of data collection concerns downloading from YouTube[4] videos and playlists. Using the Python extension *youtube_dl*[5], I bulk downloaded videos from external user-created playlists corresponding to the data I wanted to extract. I used this technique to collect the Billboard 2014 Dataset, which consists of over 200 top contemporary billboard charts in the United States. This dataset was not used in the primary experimentation but solidified notions on the harmonic nature of pop songs. After downloading these files in *mp4* format, I converted them to *wav* files and extracted their chord progressions using Chordino.

### 4.3.3 Rhapsody

*Rhapsody*[6] is a popular music streaming service that allows users to listen to a huge database of music for a low monthly fee. Already being a Rhapsody member, I used a Python script called `rapi.py`[7], an unofficial Rhapsody API that can be used to download streams given valid login credentials.

The McGill Billboard Project Dataset [6], the primary dataset used in this paper which consists of a large collection of ground-truth annotations for Billboard charts from the 1950s through the 1990s, does not contain the corresponding audio files that could be used to create extracted annotations for comparison. I wrote a Python program that looks up the title, artist, and duration of the included metadata in the dataset using the Python extension *pyechonest*[8] which queries the Echonest[9] API, a musical intelligence project that stores a massive amount of musical metadata. Some Echonest results include Rhapsody track IDs that can be used to find the corresponding stream on Rhapsody. The Python program, for every successful match to Rhapsody, downloads the corresponding audio file, converts it to the *wav* format, and extracts chord progressions, leading to the creation of the McGill Extracted Dataset.

### 4.3.4 National Anthems

The National Anthems Dataset was collected from the Wikipedia page "List of national anthems" [10] which collects open source recordings of national anthems for every country recognized by the United Nations and

[4]https://www.youtube.com/
[5]https://pypi.python.org/pypi/youtube_dl
[6]https://www.rhapsody.com/
[7]https://github.com/davekilian/rapi/blob/master/rapi.py
[8]https://github.com/echonest/pyechonest
[9]http://the.echonest.com/
[10]http://en.wikipedia.org/wiki/List_of_national_anthems

a few other states and territories not officially recognized. This dataset was used to observe the nature of harmonic comparisons outside the realm of pop music. Using Python to parse this web page, I downloaded all the audio files, which were in the *ogg* format, an open-source alternative to *mp3*, converted the files to the *wav* filetype, and ran chord extraction.

### 4.3.5 Second Hand Songs

Second Hand Songs[11] catalogues collections of *cover songs*, versions of songs recorded by someone other than the original artist. Intuitively, cover songs often have the same harmonic progressions as their original versions [20], thus Second Hand Songs provides a useful tool for harmonic comparison, ground-truthing by title of song [15]. Second Hand Songs fortunately contains YouTube links for some of its catalogued songs, so I wrote a program that iterates through the top songs on the "Most covered song" list[12] and picks out the first 50 song titles for which more than 20 YouTube versions of cover songs are listed. The first 20 YouTube videos in each of these songs is scraped according to the methods in 4.3.2, resulting in a dataset containing 1,000 songs. This dataset was used in informal experimentation evaluating the modularity [33] of its fully connected harmonic comparison graph with respect to neighborhoods defined by cover communities. A more large-scale cover song identification MIR project using SecondHandSongs is covered in [5].

[11] http://secondhandsongs.com/
[12] http://secondhandsongs.com/statistics?id=stat-most-covered-song

# Chapter 5

# Experimental Results

## 5.1 Variables Used and Notation

This chapter details the primary experiments run on the data as laid out in chapter 3. To compare chord progressions, only the Smith-Waterman algorithm ($SW$) is used, with normalization measures *raw score* and $SW_{norm}$ being tested. Gap costs ($gap_{open}$ and $gap_{ext}$) are allowed to vary from 0 through 128. The Harte ($Harte$) and Tonal Pitch Space ($TPS$) chord distance metrics are used to evaluate chord distances. Lastly, multiplication and subtraction factors $m_x$ and $m_s$ are used to round the chord distance metrics to integers with an expected value below 0. A full summary of the variables and their tested ranges is as follows:

| Variable | Notation | Values |
|---:|---|---|
| Normalization | $norm$ | $\{raw\ score, SW_{norm}\}$ |
| Gap open cost | $gap_{open}$ | $[0 - 128]$ |
| Gap extension cost | $gap_{ext}$ | $[0 - 128]$ |
| Chord Distance Metric | $C_d$ | $\{Harte, TPS\}$ |
| Chord Distance Multiplier | $m_x$ | $[1, 30]$ |
| Chord Distance Subtraction Factor | $m_s$ | $[0, 30]$ |

Figure 5.1: Summary of experimental parameters.

The two primary experiments run, *Fully Connected Pairwise Harmonic Comparison* and *Query by N-gram* are denoted FCC and QBN, respectively. The ground-truth and computationally extracted annotations from the McGill dataset (3.3) are labeled $McGill_g$ and $McGill_e$. As I was only able to automate the download of 529 songs from the McGill dataset (4.3.3), the size of $McGill_e$ ($|McGill_e|$) is 529. I limit the songs of

$McGill_g$ to only include those that were successfully chordally extracted in the same order as $McGill_e$, thus $|McGill_g| = |McGill_e|$ and the datasets are fit for comparison.

## 5.2 Optimizing Fully Connected Pairwise Harmonic Comparison

As outlined in 3.7.2, my simulated annealing optimization procedure over FCC (3.5.1) involves 1,000 iterations and a temperature gradient that decreases exponentially from 1 to 0.005 throughout the course of the algorithm. On an AMD Phenom II X4 965 3.4GHz Ubuntu quad-core desktop, FCC optimization took 9 hours and 25 seconds to run, averaging 33.9 seconds per iteration. The scoring performance of the simulated annealing algorithm can be depicted with a graph of the resulting harmonic correlational metrics updated as the iterations progress. I chart these results along with an exponential moving average[1] line in Figure 5.2.

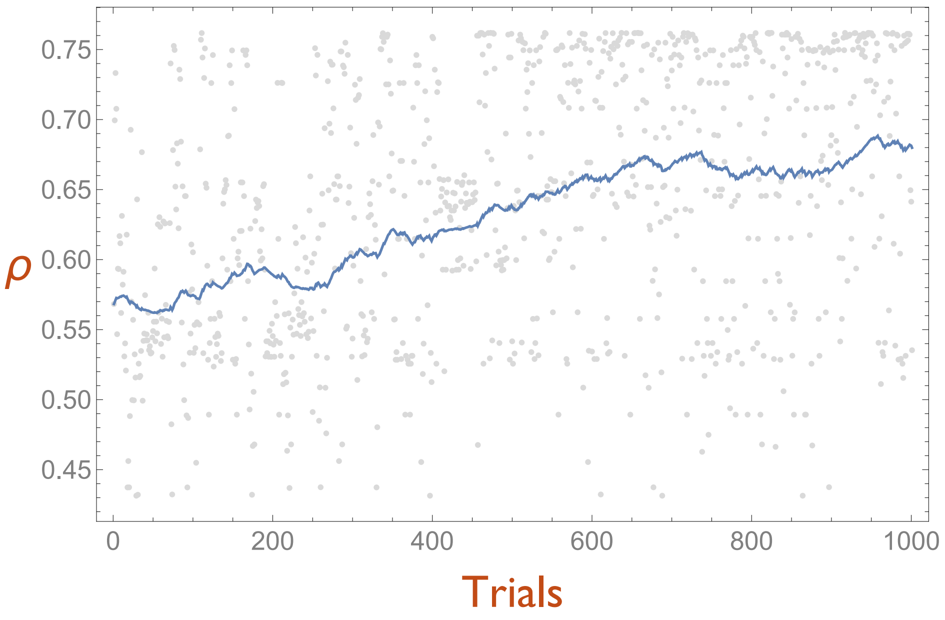

Figure 5.2: Simulated annealing performance in FCC. Each dot represents an iteration of the algorithm and correlational harmonic metric performance $\rho$. The blue line, an exponential moving average, demonstrates the increase in performance as iterations progress. FCC performance increases at a relatively constant pace as the algorithm progresses.

The maximal correlational harmonic metric $\rho$ returned by the simulated annealing was 0.761905, evidencing a strong correlation. Multiple parameter sets returned this correlation. They are as follows (Figure 5.3):

[1]An exponential moving average is a running average of a dataset that assigns the most weight to recent observations. The weights that constitute the calculation of the average decrease exponentially, tracing the observations from their beginning.

| *Iteration* | **norm** | $\mathbf{C_d}$ | $\mathbf{gap_{open}}$ | $\mathbf{gap_{ext}}$ | $\mathbf{m_x}$ | $\mathbf{m_s}$ |
|---|---|---|---|---|---|---|
| 472 | *raw* | *TPS* | 0 | 28 | 5 | 9 |
| 540 | *raw* | *TPS* | 0 | 95 | 30 | 9 |
| 636 | *raw* | *TPS* | 0 | 16 | 29 | 9 |
| 656 | *raw* | *TPS* | 0 | 4 | 1 | 9 |
| 657 | *raw* | *TPS* | 0 | 28 | 9 | 9 |
| 820 | *raw* | *TPS* | 0 | 23 | 14 | 9 |
| 824 | *raw* | *TPS* | 0 | 54 | 1 | 9 |
| 843 | *raw* | *TPS* | 0 | 0 | 18 | 9 |
| 916 | *raw* | *TPS* | 0 | 105 | 11 | 9 |
| 945 | *raw* | *TPS* | 0 | 128 | 1 | 9 |
| 976 | *raw* | *TPS* | 0 | 5 | 3 | 9 |

Figure 5.3: The tying optimal candidate parameters for FCC returning a $\rho$ of 0.761905. All parameters include the *raw* normalization and the *TPS* chord distance metric.

The optimal FCC result can be visualized by constructing a scatter plot in which each point describes a corresponding pair of ranking results from the ground-truth dataset and the extracted chord dataset ($McGill_g$ and $McGill_e$). The x- and y- axes both describe the range of rank values present. This type of correlational plot should ideally have a strong diagonal line spanning from (0,0) to the top-right corner. A scatter plot of the first of the optimal FCC parameters is provided in Figure 5.4.

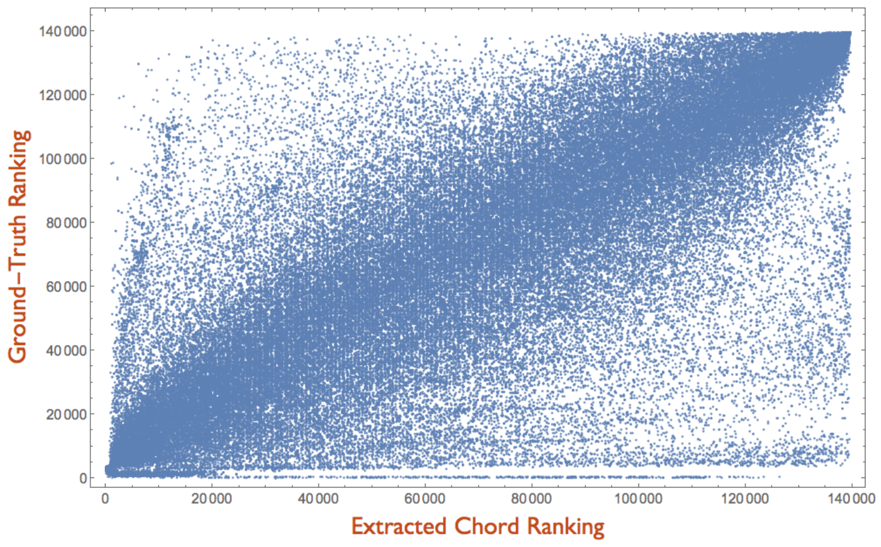

Figure 5.4: Scatter plot of the optimal fully connected comparison rankings. The correlation ($\rho$=0.76) is clearly visible through the strong diagonal band running through the results. The bottom-left corner corresponds to pairwise chord progression comparisons ranked poorly in both ground-truth and extracted datasets, while the top-left corner represents comparisons consistently ranked well.

The density of different regions of the scatter plot can be clarified with a 3-dimensional histogram in which square bins have heights corresponding to the frequency of occurrences of pairs with coordinates within the bounds of the square. A 3-dimensional histogram representation of FCC rankings is given in Figure 5.5.

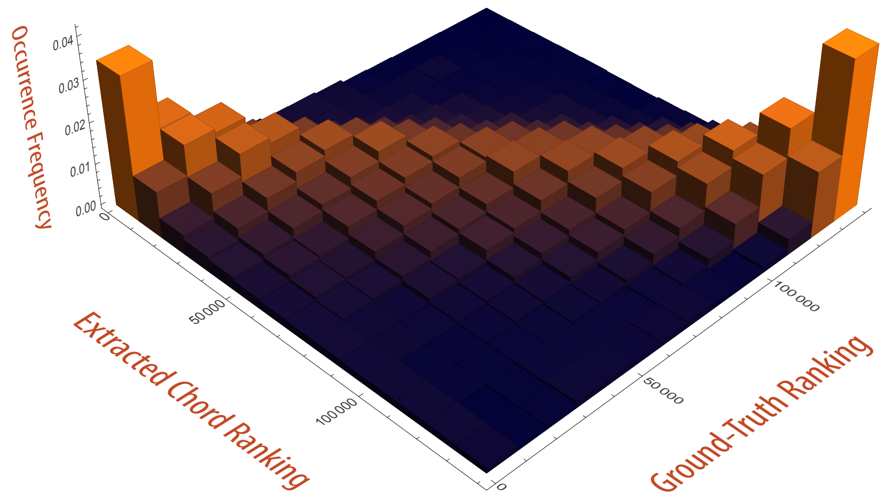

Figure 5.5: 3-dimensional histogram of the optimal fully connected comparison rankings. The correlation ($\rho$=0.76) is visible through the elevated orange diagonal band. The density of points along this band is greatest at the corners as evidenced by bin heights—this means salient strongly and weakly ranked chord progression results are most preserved by the parameters that led to this result.

The first of these optimal parameters with a $\rho$ of 0.761905 has a corresponding $\tau$ (see Section 3.6.2.2) of 0.590498. The software program Mathematica[2] can analyze the statistical significance of these rank metrics. The $p$-value of $\rho$ is $1.082 \times 10^{-26346}$, describing a next-to-zero chance that such a monotonically increasing relationship could have occurred by chance. The $p$-value of $\tau$ is $7.33 \times 10^{-23749}$. For reference, the lowest scoring FCC parameter set found in the simulated annealing run

$$\{\mathbf{norm} : raw,\ \mathbf{C_d} : Harte,\ \mathbf{gap_{open}} : 35,\ \mathbf{gap_{ext}} : 60, \mathbf{m_x} : 30,\ \mathbf{m_s} : 29\}$$

had a $\rho$ of 0.431419 ($p$-value $2.03 \times 10^{-6248}$) and a $\tau$ of 0.369491 ($p$-value $7.43 \times 10^{-5968}$).

[2]http://www.wolfram.com/mathematica/

## 5.3 Optimizing Querying by N-Gram Comparisons

Optimizing QBN (3.5.2), like FCC, involves 1,000 iterations of simulated annealing and a temperature gradient that decreases from 1 to 0.005. Unlike FCC, QBN's correlational harmonic metric is derived from an average of rank correlations ($\rho$) over 100 pregenerated n-gram queries divided equally into lengths of 4, 8, 16, and 32. These correlations are taken between the algorithmic results inputted with ground-truth chord annotations and extracted chord annotations. QBN optimization took 10 hours and 21 minutes, averaging 37.2 seconds per iteration (372 milliseconds per query). A chart depicting the scoring performance of the simulated annealing algorithm for QBN is given along with an exponential moving average line in Figure 5.6.

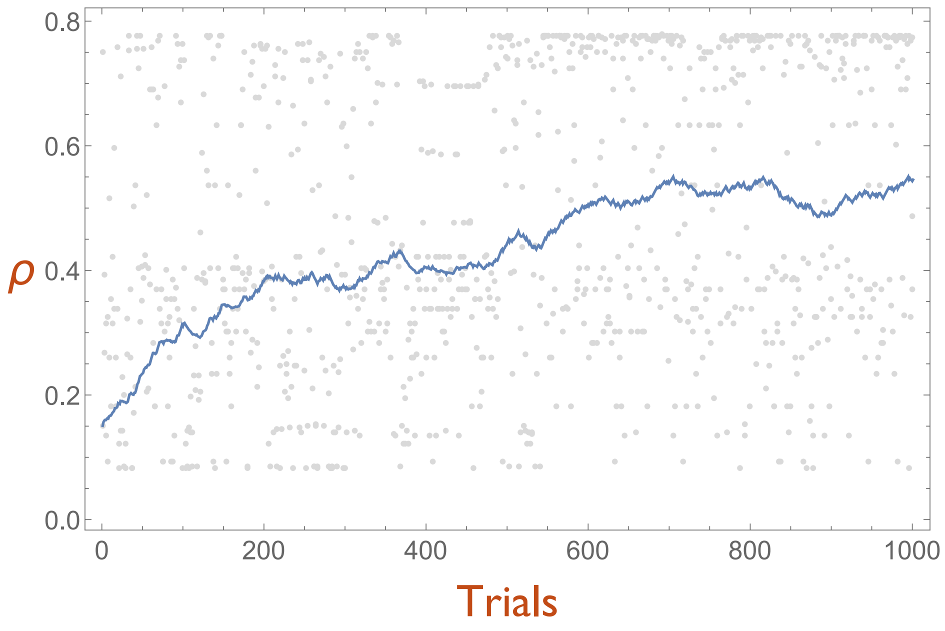

Figure 5.6: Simulated annealing performance in QBN. Each dot represents an iteration of the algorithm and correlational harmonic metric performance $\rho$. The blue line, an exponential moving average, demonstrates the increase in performance as iterations progress. QBN performance increases the most in the beginning iterations but consistently increases throughout the entirety of the optimization.

The maximal average correlational harmonic metric between all n-gram queries returned by the simulated annealing algorithm was 0.779 and occurred singularly with the following parameters:

$$\{\mathbf{norm} : SW_{norm},\ \mathbf{C_d} : TPS,\ \mathbf{gap_{open}} : 1,\ \mathbf{gap_{ext}} : 82, \mathbf{m_x} : 1,\ \mathbf{m_s} : 10\}$$

This optimal QBN result can be visualized as a series of scatter plots corresponding to the output rankings of each query n-gram. Rather than use a 3-dimensional histogram to show the density of the scatter plots, a more compact 2-dimensional density plot can be used in which colors range according to a color scale and represent the same information as a 3-dimensional histogram bin without visual depiction of height. The resulting scatter plots and density plots can be grouped by length of n-gram. These results are shown in Figure 5.7.

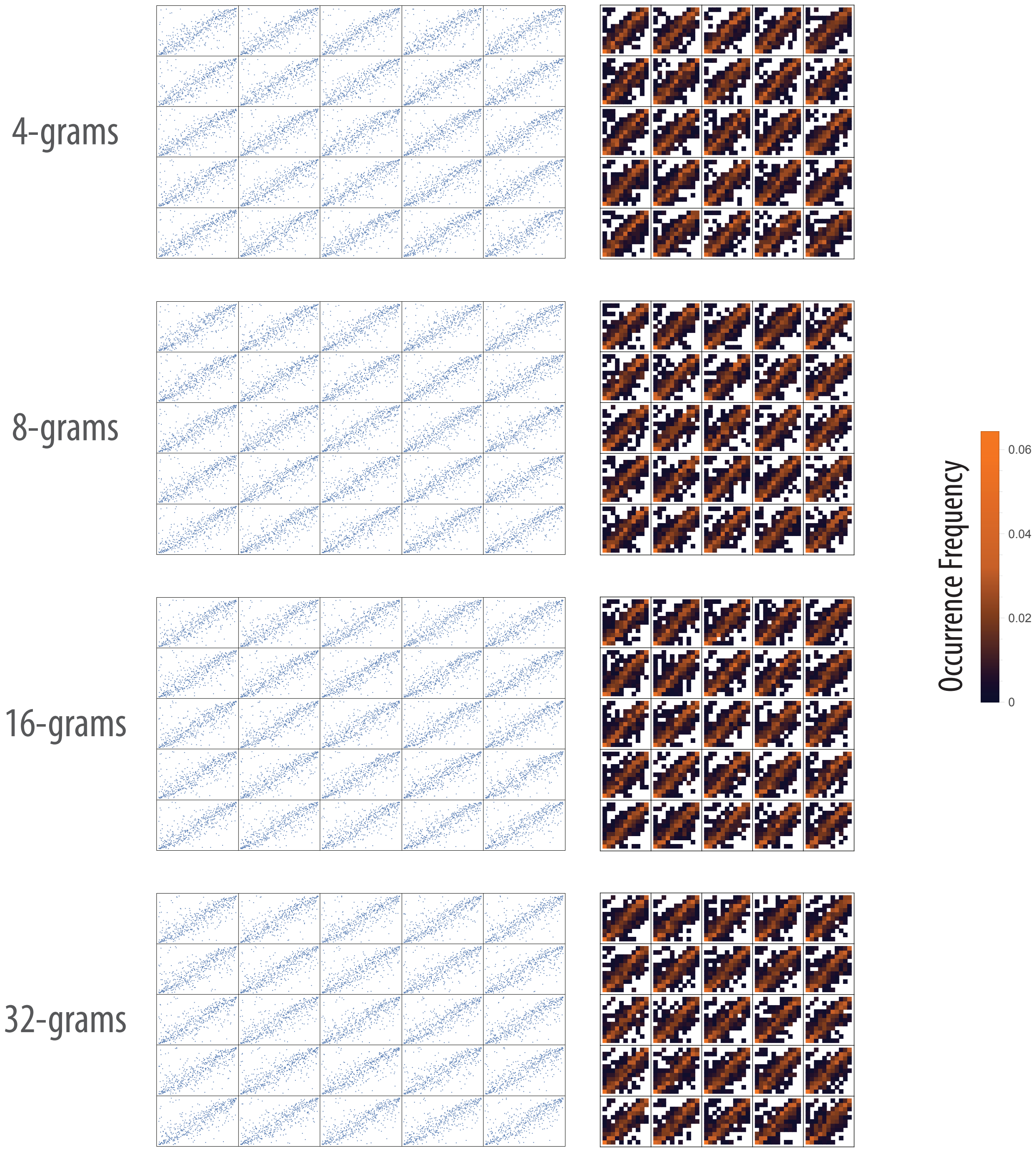

Figure 5.7: Scatter plots and density plots of the optimal n-gram query comparison rankings. Each of the 100 random queries used are distributed equally into 4-, 8-, 16-, and 32-grams. Each n-gram length contains 5x5 subplots showing rank correlation scatter and density plots of the individual query's chord progression rankings. All the density plots are normalized to the same color scale, corresponding with frequency of bin occurrence. The similar shape of all the scatter and density plots with strong diagonal bands shows that each query performed well, maintaining a significant positive correlation of output comparison result between inputted ground-truth (x-axis on each subplot) and human-annotated (y-axis on each subplot) datasets (average $\rho = 0.78$).

## 5.4 Parameter Optimization

The harmonic retrieval tasks presented in this paper, FCC and QBN, rely on a common set of parameters $p$. As the optimization algorithms only surveyed a limited realm of the search space, generalizations on effective values for the parameter set cannot be fully founded. Nonetheless, it can still be useful to future experimentation to detail average correlational harmonic metric values associated with ranges of parameter values. These inform Smith-Waterman algorithm inputs that seem to work well. Future experimentation is needed to form more definitive conclusions.

$norm$ and $C_d$ are the variables that perhaps change the nature of the Smith-Waterman function the most fundamentally. Average output correlational harmonic metric values for inputted choices of $norm$ and $C_d$ based on parameter movements in the simulated annealing optimization algorithm are as follows for FCC:

| | | $\mathbf{C_d}$ | |
|---|---|---|---|
| | | $Harte$ | $TPS$ |
| **norm** | $raw$ | 0.59 | <u>0.68</u> |
| | $SW_{norm}$ | 0.56 | 0.62 |

where maximum values are underlined. Correlational harmonic metric values for QBN based on the same set of parameters are:

| | | $\mathbf{C_d}$ | |
|---|---|---|---|
| | | $Harte$ | $TPS$ |
| **norm** | $raw$ | 0.40 | 0.49 |
| | $SW_{norm}$ | 0.35 | <u>0.53</u> |

According to these observational resutls, TPS outperforms the Harte chord distance metric in both experiments in terms of maximizing correlation.

$gap_{open}$ and $gap_{ext}$ take a wider range of values, and it is thus more useful to look at variable ranges and their average outputs. The following table shows average FCC and QBN correlational harmonic metrics corresponding to ranges of gap variable values:

|  | FCC |  | QBN |  |
|---|---|---|---|---|
| Range | $\mathbf{gap_{open}}$ | $\mathbf{gap_{ext}}$ | $\mathbf{gap_{open}}$ | $\mathbf{gap_{ext}}$ |
| 0 | 0.71 | 0.64 | 0.69 | 0.49 |
| $\leq 8$ | 0.67 | 0.62 | 0.46 | 0.49 |
| $> 8$ | 0.61 | 0.64 | 0.34 | 0.47 |

These results suggest that gap opening penalties of 0 influence higher correlational harmonic metric score. One hypothesis for this behavior is that extracted chord data may frequently be offset by a single chord from corresponding ground-truth data due to an error in extraction. A gap opening penalty is the most forgiving in these cases.

Lastly, I chart which rounded chord metric variables ($m_x$ and $m_s$) produced the highest average correlation harmonic metric scores. In FCC:

|  |  | $\mathbf{m_x}$ |  |  |
|---|---|---|---|---|
|  |  | $\geq 0$ *and* $< 10$ | $\geq 10$ *and* $< 20$ | $\geq 20$ |
| $\mathbf{m_s}$ | $\geq 0$ *and* $< 10$ | 0.67 | 0.64 | 0.69 |
|  | $\geq 10$ *and* $< 20$ | 0.68 | 0.65 | 0.65 |
|  | $\geq 20$ | 0.58 | 0.58 | 0.57 |

And in QBN:

|  |  | $\mathbf{m_x}$ |  |  |
|---|---|---|---|---|
|  |  | $\geq 0$ *and* $< 10$ | $\geq 10$ *and* $< 20$ | $\geq 20$ |
| $\mathbf{m_s}$ | $\geq 0$ *and* $< 10$ | 0.62 | 0.51 | 0.65 |
|  | $\geq 10$ *and* $< 20$ | 0.51 | 0.43 | 0.49 |
|  | $\geq 20$ | 0.39 | 0.38 | 0.30 |

These results are both consistent in assigning higher correlational harmonic metric scores to large multiplication factors and small subtraction factors. A possible explanation for this behavior is that these factor choices result in the highest Smith-Waterman chord distance expected values. Though this expected value is ensured to be negative by forcing parameter choices such that this is the case, a value close to 0 will more frequently match chords positively by chance and result in longer local alignment scores that resemble global alignment scores. It is possible that global sequence alignment techniques used in FCC and QBN have strong correlational harmonic metric scores. Further research in global sequence alignment could present promising correlational harmonic metric results.

# Chapter 6

# Conclusion

This project explored the use and creation of a new class of metrics in MIR—*correlational metrics*. Through the lens of harmony, I endeavored to show that traditional MIR tasks can be configured with established algorithms such that they perform as effectively with computationally extracted chord data as they do with human-annotated ground-truth chord data. There are few research-backed datasets of the latter class of chord annotations, whereas extracted chord data can be automatically generated from audio files. The implications of my methods suggest that practical MIR systems can be constructed and optimized to work without the guide of human ground-truthing, infinitely expanding the possible realm of chord datasets to harmonically compare.

In the process of constructing my system, I modified existing hyper-efficient sequence alignment algorithms traditionally used in bioinformatics to the alphabet of musical chords. I developed a bitwise representation of harmony that facilitated rapid computation, and my end methods surpassed the speeds of similar documented techniques in other research by a factor of over 100x. These speed-ups further encourage the use of localized sequence alignment techniques in large-scale MIR systems.

In its current form, my research is limited to Western harmonies, and more specifically, pop songs from the 1950s onwards. There is ample room for further experimentation. Many other features could have been investigated, from those directly supplemental to harmony, such as chord duration and melody, to external factors, such as song popularity or artist. Incorporating and testing more chord distance metrics and different parameters would enrich the range of my work.

One particular facet of unexplored MIR research that could substantively advance understandings of harmony is *multiple sequence alignment*. Multiple sequence alignment methods are similar to traditional sequence alignment methods such as the Smith-Waterman algorithm; they differ in that they can be utilized to compare more than two sequences simultaneously. In the context of chord progression sequences, these methods can be used to construct phylogenetic musical trees and run evolutionary analyses. I minimally explored these techniques, though a proper implementation would fit well in a system to assess correlational harmonic metrics.

As modern theorist Arnold Schönberg muses in his *Theory of Harmony*, "the Evolution of no other art is so greatly encumbered by its teachers than that of music" [41]. Only by detaching strict theoretical underpinnings of ground-truth from harmonic MIR tasks can practice precede theory and evolve computational music understanding. Correlational harmonic metrics provide the tools for liberating a nascent field bound by its subjective principles.

# Bibliography

[1] Elie Adam, Elie El Nouné, and Yasmina Yared. *A System for Music Similarity Search Based on Harmonic Content.*

[2] Julien Allali et al. "Local transpositions in alignment of polyphonic musical sequences". In: *String Processing and Information Retrieval.* Springer. 2007, pp. 26–38.

[3] Willi Apel. *Harvard Dictionary of Music, 2nd Revised and Enlarged Edition*. Belknap Press, 1969. ISBN: 0435810006.

[4] S. Bechhofer et al. "Computational Analysis of the Live Music Archive". In: *Presented at the 15th International Society of Music Information Retrieval (ISMIR) Conference late-breaking workshop, Oct 27-31, 2014, Taipei, Taiwan.* 2014.

[5] Thierry Bertin-Mahieux and Daniel PW Ellis. "Large-scale cover song recognition using hashed chroma landmarks". In: *Applications of Signal Processing to Audio and Acoustics (WASPAA), 2011 IEEE Workshop on.* IEEE. 2011, pp. 117–120.

[6] John Ashley Burgoyne, Jonathan Wild, and Ichiro Fujinaga. "An Expert Ground-Truth Set for Audio Chord Recognition and Music Analysis". In: *Proceedings of the 12th International Society for Music Information Retrieval Conference.* http://ismir2011.ismir.net/papers/OS8-1.pdf. Miami (Florida), USA, 2011, pp. 633–638.

[7] Chris Cannam et al. *MIREX 2013 entry: Vamp plugins from the centre for digital music*. 2013.

[8] Vladimír Černỳ. "Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm". In: *Journal of optimization theory and applications* 45.1 (1985), pp. 41–51.

[9] Kun-Mao Chao, William R Pearson, and Webb Miller. "Aligning two sequences within a specified diagonal band". In: *Computer applications in the biosciences: CABIOS* 8.5 (1992), pp. 481–487.

[10] Bas De Haas, Remco Veltkamp, and Frans Wiering. "Tonal Pitch Step Distance: A Similarity Measure for Chord Progressions". In: *Proceedings of the 9th International Conference on Music Information Retrieval*. http://ismir2008.ismir.net/papers/ISMIR2008_252.pdf. Philadelphia, USA, 2008, pp. 51–56.

[11] W Bas De Haas, Frans Wiering, and Remco C Veltkamp. "A geometrical distance measure for determining the similarity of musical harmony". In: *International Journal of Multimedia Information Retrieval* 2.3 (2013), pp. 189–202.

[12] W Bas De Haas et al. "Comparing approaches to the similarity of musical chord sequences". In: *Exploring Music Contents*. Springer, 2011, pp. 242–258.

[13] Persi Diaconis and Ronald L Graham. "Spearman's footrule as a measure of disarray". In: *Journal of the Royal Statistical Society. Series B (Methodological)* (1977), pp. 262–268.

[14] "Harmony and Melody". In: *Dwight's Journal of Music: A Paper of Art and Literature, Volumes 23-24*. Ed. by John Sullivan Dwight. Vol. 3. 1864.

[15] W Bas de Haas et al. "Comparing Harmonic Similarity Measures". In: *Málaga (Spain)* (2010), p. 299.

[16] Pierre Hanna, Matthias Robine, and Thomas Rocher. "An alignment based system for chord sequence retrieval". In: *Proceedings of the 9th ACM/IEEE-CS joint conference on Digital libraries*. ACM. 2009, pp. 101–104.

[17] Pierre Hanna et al. "Improvements of alignment algorithms for polyphonic music retrieval". In: *Computer Music Modeling and Retrieval 2008*. 2008, pp. 244–251.

[18] Christopher Harte. "Towards automatic extraction of harmony information from music signals". PhD thesis. Department of Electronic Engineering, Queen Mary, University of London, 2010.

[19] Marc Hirsh. "Striking a chord". In: *The Boston Globe* (2008).

[20] Maksim Khadkevich and Maurizio Omologo. "Large-scale cover song identification using chord profiles". In: *Proceedings of the 14th International Society for Music Information Retrieval Conference*. http://www.ppgia.pucpr.br/ismir2013/wp-content/uploads/2013/09/67_Paper.pdf. 2013.

[21] Maksim Khadkevich and Maurizio Omologo. "Time-frequency reassigned features for automatic chord recognition". In: *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference*. IEEE. 2011, pp. 181–184.

[22] Scott Kirkpatrick et al. "Optimization by simulated annealing". In: *Science* 220.4598 (1983), pp. 671–680.

[23] Arthur Koestler. *The Sleepwalkers*. Grosset & Dunlap, New York, 1959.

[24] Kyogu Lee. "A System for Acoustic Chord Transcription and Key Extraction from Audio Using Hidden Markov Models Trained on Synthesized Audio". PhD thesis. 2008. ISBN: 978-0-549-49012-8.

[25] Kyogu Lee and Malcolm Slaney. "Acoustic chord transcription and key extraction from audio using key-dependent HMMs trained on synthesized audio". In: *Audio, Speech, and Language Processing, IEEE Transactions on* 16.2 (2008), pp. 291–301.

[26] Fred Lerdahl. "Tonal pitch space". In: *Music Perception* (1988), pp. 315–349.

[27] VI Levenshtein. "Binary Codes Capable of Correcting Deletions, Insertions and Reversals". In: *Soviet Physics Doklady*. Vol. 10. 1966, p. 707.

[28] Matthias Mauch and Simon Dixon. "Approximate note transcription for the improved identification of difficult chords". In: *in Proc. 11th Int. Soc. Music Inf. Retrieval Conf. (ISMIR)*. 2010, pp. 135–140.

[29] Matthias Mauch and Simon Dixon. "Simultaneous estimation of chords and musical context from audio". In: *Audio, Speech, and Language Processing, IEEE Transactions on* 18.6 (2010), pp. 1280–1289.

[30] Matthias Mauch, Katy Nol, and Simon Dixon. "Using musical structure to enhance automatic chord transcription". In: *Proc. ISMIR*. 2009.

[31] Matthias Mauch et al. *OMRAS2 Metadata Project 2009*. Tech. rep. 2009.

[32] Matt McVicar et al. "Automatic chord estimation from audio: A review of the state of the art". In: *Audio, Speech, and Language Processing, IEEE/ACM Transactions on* 22.2 (2014), pp. 556–575.

[33] Mark EJ Newman. "Modularity and community structure in networks". In: *Proceedings of the National Academy of Sciences* 103.23 (2006), pp. 8577–8582.

[34] Mitsunori Ogihara and Tao Li. “N-Gram Chord Profiles for Composer Style Representation.” In: *ISMIR*. Ed. by Juan Pablo Bello, Elaine Chew, and Douglas Turnbull. Dec. 28, 2009, pp. 671–676. ISBN: 978-0-615-24849-3. URL: http://dblp.uni-trier.de/db/conf/ismir/ismir2008.html#OgiharaL08.

[35] Mitsunori Ogihara and Tao Li. “N-Gram Chord Profiles for Composer Style Representation.” In: *ISMIR*. 2008, pp. 671–676.

[36] Laurent Oudre, Yves Grenier, and Cédric Févotte. “Template-Based Chord Recognition: Influence of the Chord Types”. In: (2009).

[37] Jean-Philippe Rameau. *Treatise on Harmony*. Trans. by Philip Gossett. Dover Publications, New York, 1971.

[38] Thomas Rocher et al. “A Survey of Chord Distances With Comparison for Chord Analysis”. In: *International Computer Music Conference (ICMC)*. 2010, pp. 187–190.

[39] Heinrich Schenker. *Harmony*. Ed. by Oswald Jonas. The University of Chicago Press, Chicago, 1954.

[40] Arnold Schoenberg. *Structural Functions of Harmony*. Ed. by Leonard Stein. Ernest Benn Limited, London, 1969.

[41] Arnold Schoenberg. *Theory of Harmony*. Trans. by Roy E. Carter. Faber and Faber, London, 1978.

[42] Tibor Serly. *A Second Look at Harmony*. Samuel French, New York, 1964.

[43] Frederick G. Shinn. *A Method of Teaching Harmony*. The Vincent Music Company, London, 1904.

[44] Temple F Smith and Michael S Waterman. “Identification of common molecular subsequences”. In: *Journal of molecular biology* 147.1 (1981), pp. 195–197.

[45] Temple F Smith, Michael S Waterman, and Christian Burks. “The statistical distribution of nucleic acid similarities”. In: *Nucleic Acids Research* 13.2 (1985), pp. 645–656.

[46] L. H. Southard. *Course of Harmony*. George P. Reed & Company, Boston, 1855.

[47] James Tenney. *A History of ‘Consonance’ And ‘Dissonance’*. Excelsior Music Publishing Company, New York, 1988.

[48] Ernst Terhardt. “Pitch, consonance, and harmony”. In: *The Journal of the Acoustical Society of America* 55.5 (1974), pp. 1061–1069.

[49] Yushi Ueda et al. “HMM-based approach for automatic chord detection using refined acoustic features”. In: *Acoustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference*. IEEE. 2010, pp. 5518–5521.

[50] Alexandra L Uitdenbogerd and Justin Zobel. “An architecture for effective music information retrieval”. In: *Journal of the American Society for Information Science and Technology* 55.12 (2004), pp. 1053–1057.

[51] Wikipedia. *List of songs containing the I–V–vi–IV progression — Wikipedia, The Free Encyclopedia*. [Online; accessed 14-February-2015]. 2015. URL: http://en.wikipedia.org/wiki/List_of_songs_containing_the_I%E2%80%93V%E2%80%93vi%E2%80%93IV_progression.

[52] Mengyao Zhao et al. “SSW Library: An SIMD Smith-Waterman C/C Library for Use in Genomic Applications”. In: *PLoS ONE* 8.12 (2013). Ed. by Leonardo Mariño-Ramírez, e82138. DOI: 10.1371/journal.pone.0082138. URL: http://dx.doi.org/10.1371/journal.pone.0082138.