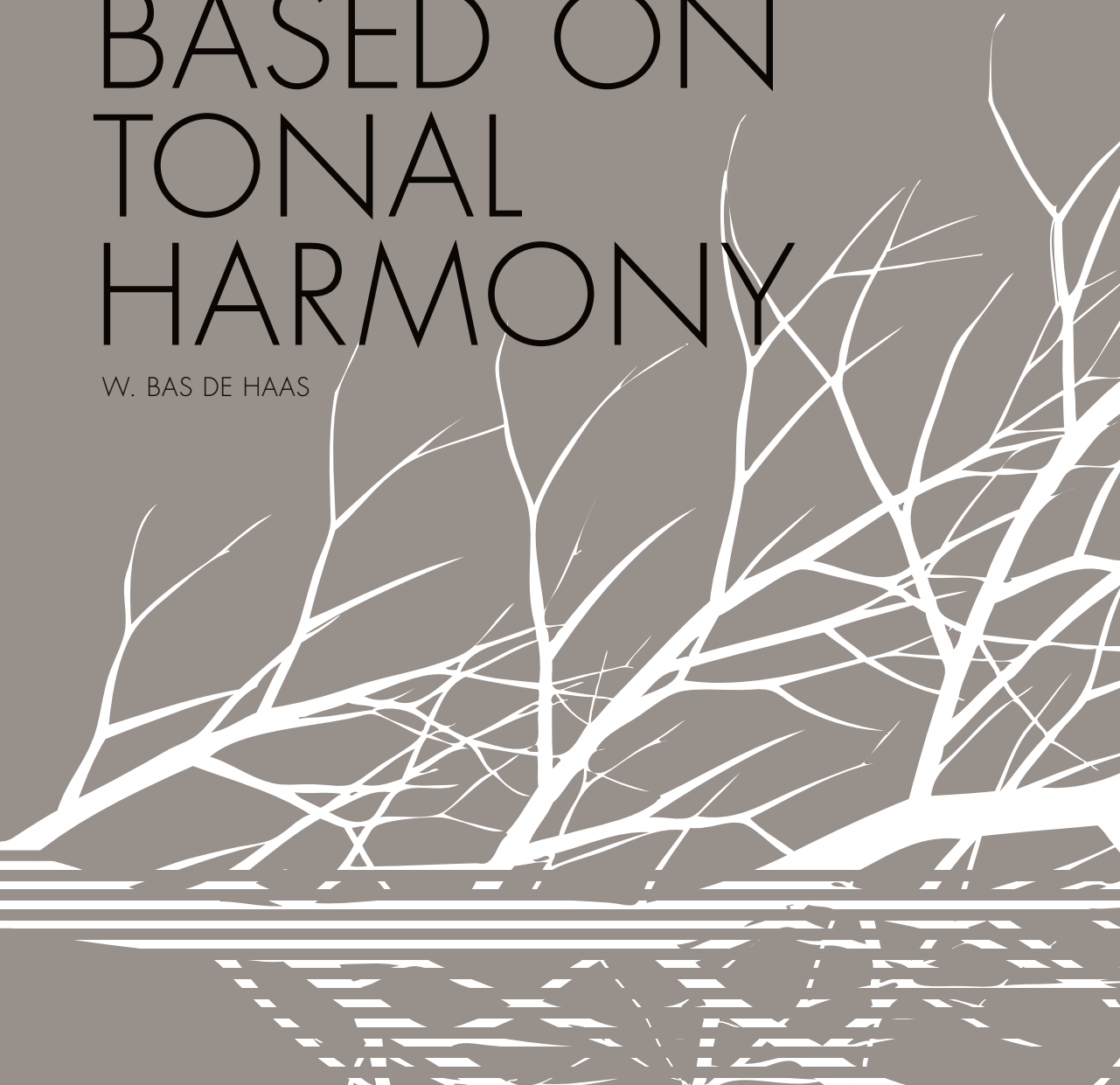
MUSIC INFORMATION RETRIEVAL BASED ON TONAL HARMONY

W. BAS DE HAAS



Music information retrieval based on tonal harmony

W. Bas de Haas

ISBN 978-90-393-5735-4

Cover design by 945 Ontwerp, http://www.945-ontwerp.nl/.

Typeset in LaTeX
Printed in the Netherlands by Koninklijke Wöhrmann, http://nl.cpibooks.com/.

The research presented in this thesis has been partially funded by the Dutch ICES/KIS III Bsik project MultimediaN.

Music information retrieval based on tonal harmony

Zoeken naar muziek op basis van tonale harmonie
(met samenvatting in het Nederlands)

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Universiteit Utrecht
op gezag van de rector magnificus, prof.dr. G.J. van der Zwaan,
ingevolge het besluit van het college voor promoties
in het openbaar te verdedigen op
dinsdag 28 februari 2012
des middags te 2.30 uur

door

Willem Bas de Haas

geboren op 4 augustus 1980
te Groningen

Promotor: Prof.dr. R.C. Veltkamp
Co-promotor: Dr. F. Wiering

Contents

Chapter 1

Introduction

MUSIC, as vivid, diverse, culturally dependent, highly subjective and sometimes even intangible art form, might not be the first thing associated with computer science. Nonetheless, the last four years I have been in a privileged position that allowed me to treat my computer first and foremost as a musical machine; the outcome of this endeavour is the doctoral dissertation you are now holding. It goes without saying that computers are suitable for playing music, creating music, and can even aid in performing music. However, listening, understanding, and appreciating music is something rarely associated with straightforward number crunching. In this thesis I will argue that not only can we make computers grasp a glimpse of the meaning of music, but that music is much more regular and structured than one perhaps would have initially guessed, and that in particular Western tonal harmony can be formalised at least up to a considerable extent.

Although the investigation of computational models of music is interesting in itself because it helps us to understand how human listeners experience music, musical models are also important for organising large collections of music. The research area concerned specifically with this task, and also the main area that this thesis contributes to, is the area of Music Information Retrieval (MIR) research. From its tentative beginnings in the 1960s, MIR has evolved into a flourishing "multidisciplinary research endeavour that strives to develop innovative content-based searching schemes, novel interfaces, and evolving networked delivery mechanisms in an effort to make the world's vast store of music accessible to all" (Downie 2003). This store is indeed vast, especially in the digital domain. The amount of albums stored by iTunes, Spotify, or any other digital music service exceeds the number of albums that used to be available in your local record shop by a substantial factor. It would take years if not lifetimes of continuous listening to play through all these. All of this music is accessible in the sense that it can be reached within a few mouse clicks, supposed you know where to click. This is where the challenge begins, for under many circumstances listeners do not have the necessary information to be able to do so. Consider the following situation.

> As a guitar player, I am quite fond of Robben Ford, a guitar player who, in my view, takes blues improvisation to the next level by combining it with various elements of jazz[1]. I find his albums from the late eighties and begin nineties especially delectable. Even though some pieces remain appealing even after hundreds of listening experiences, I cannot suppress the desire for something new yet similar in certain aspects, e.g. instrument, groove, ensemble, emotional intensity, etc. Unfortunately, Robben Ford will not be able to fulfil this need at short notice: it is unclear when he will release new material. Also, in my view, his recent work does not exhibit the same vitality that his older work does. However, since the world provides for numerous excellent, well-schooled, creative guitar players, other artists might be capable of satisfying my aching desire. Nowadays, their music will very probably be readily available on-line, e.g. via iTunes, Amazon, Facebook, or the like. The only problem is actually finding it, especially when the artists are not among the ones best known to the general public. In other words, I do not know the name of the artist I am looking for, I do not know where (s)he comes from, let alone the title of a piece. Hence, submitting a textual query to a search engine like Google approximates random search. Clearly, search methods are needed that are specifically designed for musical content.

This is where MIR aims to provide a helping hand. However, delivering the right music—or, in the terminology of Information Retrieval, relevant music—by means of computational methods is not a solved problem at all. What music is relevant to such common but complex user needs as the one described above, depends on factors such as the user's taste, expertise, emotional state, activity, and cultural, social, physical and musical environment. Generally, MIR research abstracts from these problems by assuming that unknown relevant music is in some way similar to music known to be relevant for a given user or user group. As a consequence, the notion of *music similarity* has become a central concept in MIR research. It is not an unproblematic concept, though.

Similarity has generally been interpreted in MIR to be a numerical value for the resemblance between two musical items according to one or more observable features. For audio items, such features could be tempo or various properties of the frequency spectrum; for encoded notation, pitch, duration, key and metre are example features. The first problem to solve is to extract such features from the musical data using ap-

[1] see for a particular fine example: `http://www.youtube.com/watch?v=HvFsvmoUsnI` (An excerpt of a 1989 recording of an Italian TV show in which Robben Ford plays *Ain't got nothin' but the blues*, accessed 18 January 2012).

propriate computational methods. The next problem is to create effective methods for calculating similarity values between items. Based on these values, items can then be ranked or clustered, for which again various methods can be devised. Finally, these methods need to be evaluated individually and in combination. This is usually done by comparing their result to the ideal solution, the ground truth produced by humans manually (or aurally) performing the same task. Usually, the computational result is far from perfect. Indeed, there seems to be a glass ceiling for many MIR tasks that lies (for audio tasks) around 70% accuracy (Aucouturier and Pachet 2004; Wiggins et al. 2010, p. 234). Many MIR researchers have concluded from this that not all information is in the data, and that domain knowledge about how humans process music needs to be taken into account.

This is true in particular for the specific kind of music similarity studied in this thesis: harmonic similarity. What musical harmony is and how it relates to harmonic similarity is covered in Section 1.4. However, we will first present a brief outline of MIR (Section 1.1) and elaborate some of the issues of data-oriented MIR approaches in more depth (Section 1.2). Even though the primary goal of MIR is not to develop theories and models of music that contribute to a better understanding of music, we will argue that it is not possible to develop effective MIR systems without making these systems musically aware by importing knowledge from music theory and music cognition in Section 1.3. Finally, we will sketch the outline of this dissertation in Section 1.5 and summarise its main contributions.

1.1 A brief outline of music information retrieval

The MIR research community is shaped to a large extent by the International Society for Music Information Retrieval (ISMIR[2]; Downie et al. 2009) and its yearly conference. Even a quick glance through the open access ISMIR proceedings[3] shows that MIR is a very diverse research area. Here we can only present a condensed overview: for more comprehensive overviews we refer to (Orio 2006; Casey et al. 2008).

Within the community at least three views on musical data coexist; that is, music as represented by metadata (originating from the library science subcommunity), by encoded notation (from musicology) and by digital audio (from digital signal processing). Computational search and classification methods have been designed independently from each of these viewpoints, although occasionally viewpoints have been combined. In addition, much research goes into providing infrastructural services for MIR meth-

[2] http://www.ismir.net

[3] http://www.ismir.net/proceedings/

ods and systems, for example research in feature extraction, automatic transcription and optical music recognition. Automatic analysis of music tends to be subsumed under MIR as well, for example the creation of analytical tools, performance analysis and quantitative stylistics. Much research is directed towards visualisation, modelling mood and emotion, interfaces for engaging with music (e.g. playing along, Karaoke), playlist generation, collaborative tagging and industrial opportunities. Despite this diversity, there seems to be a strong awareness of coherence within the MIR community, which can largely be explained by a shared ideal, very similar to Downie's quoted above, of universal accessibility of music and by a common commitment to empirical observation, modelling and quantitative evaluation.

Today's achievements and advances in MIR are probably best illustrated by the Music Information Retrieval EXchange (MIREX[4]; Downie 2008; Downie et al. 2010). MIREX is a community-based initiative that provides a framework for objectively evaluating MIR related tasks. Each year during the ISMIR conference, the results of the evaluation of around 15-20 different tasks are presented, each with on average 8-9 submissions. Example tasks include Audio Beat Tracking, Audio Key Detection, Structural Segmentation, Query by Singing/Humming and Symbolic Melodic Similarity. Submissions that have been evaluated for the last task over the years include geometric, sequence alignment, graph-based, n-gram and implication-realisation based approaches. Considerable progress has been made in most tasks since 2005. Especially the transcription of low-level audio features into higher-level musical symbols such as beats, notes, chords and song structure has seen substantial improvement.

Nevertheless, moving from research prototypes to industry-strength applications is difficult and the number of functioning systems that are actually capable of solving issues like the one raised in the previous section is very limited. To get a somewhat sensible answer one's best bet is still to use services such as Last.fm[5] and Pandora[6] , which are based on very rich metadata (including social tagging). Although the retrieval performance of these services is among the best currently available on the web, we are convinced that integrating content-based search methods into these will improve their performance. However, to realise this potential, a major step beyond the dominant data-oriented approach needs to be taken.

[4] `http://www.music-ir.org/mirex/wiki/MIREX_HOME`
[5] `http://www.last.fm/`
[6] `http://www.pandora.com`

1.2 The limitations of data-oriented MIR

Before we present our critical notes on the machine learning paradigm, we would like to stress that machine learning is an important area of research with many useful, practical and theoretical applications in MIR. However, we will argue here that a purely data-oriented approach is not sufficient for meaningful music retrieval, i.e. music retrieval that steps beyond mere similarity of content features in order to deliver music that actually makes sense to the user 'here and now', and contributes to his/her experience, enjoyment and/or understanding of music (Wiering 2009). We distinguish the following six limitations to data-oriented MIR.

Ground-truth and data. Probably the greatest weakness of data-oriented approaches is the data itself, or more precisely, the lack of it. To be able to train a supervised learning algorithm a very substantial amount of data has to be annotated—in the case of MIR often by musical experts—and it is known that the larger the amount of data is, the better the algorithm performs (Harris-Jones and Haines 1998). Obtaining such ground-truth data is a costly and time-consuming enterprise, which moreover is often frustrated by copyright issues. In practice, therefore, such sets tend to be small. In addition, it is hard to generalise annotations and similarity judgements of a small number of experts collected in an experimental setting to the much larger population of possible end-users studying or enjoying music in 'ecological' circumstances.

Danger of overfitting. Supervised learning algorithms are all based on optimising the parameters of a model in such way that the difference (error) between the predictions of this model and the expert annotations is minimised. Obviously, the more flexible a model is, i.e. the more adjustable parameters it has, the better it can fit the data. As a consequence, a flexible model will often have a larger prediction error on other data sets than a less flexible model because it was trained to explain the noise in the training set as well. This process is often referred to as overfitting (e.g., Pitt and Myung 2002; Bishop 1995, p. 332). In MIR, the specific issue is that there are only a few annotated data sets that can be used for testing trained systems. It is therefore hard to assess the claimed retrieval results: it is often unclear if these systems present an improvement that can be generalised to other data sets, or if they are merely overfitting the currently available data sets.

Curse of dimensionality. Music is a complex phenomenon; therefore a considerable number of musical features need to be taken into account at any given point in time. For example, an MIR system may need information about simultaneously sounding notes, their timbre, intonation, intensity, harmonic function, and so on. As a result the input vector, i.e. the list of numerical values representing these features, is often

high-dimensional. This introduces the so-called curse of dimensionality (Bishop 1995, p. 7): the problem of the exponential increase in volume of the search space caused by adding extra dimensions to the input data, whereas the data itself becomes very sparse in this space. The amount of training data also needs to increase exponentially in order to attain a model with the same precision as a corresponding low-dimensional one.

Neglecting time. One of the most distinctive features of music is that it evolves over time: there is no such thing as music without time. The fundamental role of time is illustrated by the fact that the perception of a musical event is largely determined by the musical context in which it occurs, i.e. what the listener has heard before (e.g., Schellenberg 1996; Krumhansl 2001; Desain and Honing 1993). A significant number of data-oriented approaches completely disregard this fact. For example, when dealing with audio data, a common paradigm is to split an audio file up into small (overlapping) windows. Subsequently, a feature vector is created for each window, which contains characteristics of the signal (for example chroma features, MFCCs). These feature vectors are inputted into a classifier for training and the temporal order of the feature vectors, and thus the notion of musical time, is lost in the process. In a sense this resembles analysing the story in a movie while randomly mixing all the individual frames.

Nothing to learn? Another drawback of most data-oriented approaches is that it is hard to grasp what a system has actually learned. For instance, it is quite hard to interpret the parameters of a neural network or a hidden Markov model. This makes it difficult to infer how a system will respond to new unseen data. After all, how would one know whether the machine learning process did not overfit the system to the data? Moreover, for music researchers it is impossible to learn anything from the predictions of the model, because the model itself is difficult to interpret in humanly understandable, let alone musical terms.

Not all information is in the data. Last, but not least, music only becomes music in the mind of the listener. Hence, only part of the information needed for sound judgement about music can be found in the musical data. An important kind of information that is lacking in the data is the information about which bits of data are relevant to the musical (search) question and which bits are not, because this is often not clear from the statistical distributions in the data. For instance, in a chord sequence not every chord is equally important, e.g. removing passing chords or secondary dominants is far less problematic than leaving out a dominant or tonic chord, and a harmonic analysis of the piece is needed to identify the important chords. Similarly, most musically salient events occur at strong metrical positions, and a model is needed to determine where these positions are.

Furthermore, the experience of musical events strongly depends on the context in which the event occurs. This context depends on cultural, geographical and social factors, and specific user taste. One needs only to imagine playing a piece that is appropriate in a church at a dance party (or vice versa) to realise this. It is known that musically trained as well as untrained listeners have extensive knowledge about music (Bigand and Poulin-Charronnat 2006; Honing and Ladinig 2009). Herein, exposure to music plays a fundamental role. In other words, humans acquire a significant amount of knowledge about music from the data, and as for each human this exposure has been different, the outcome is bound to be different as well.

An often-heard argument in favour of machine learning is that, if humans can learn it, machines must be capable of learning it as well. However, we argue that, even in theory and under perfect circumstances, a data-oriented approach to MIR is not sufficient. Given a very complex model with enough degrees of freedom, similar to the human brain; thousands of hours of music; and, most importantly, the required relevance feedback, one still needs a model that captures the (music) cognitive abilities similar to those of a newborn, and supports the acquisition of musical skills. The reality is that models of such complexity do not exist, nor is there any certainty that they will come into existence soon, let alone that they can be learned from a few songs. Therefore, in practice purely data-oriented approaches have considerable limitations when dealing with musical information.

1.3 A knowledge-based alternative

Music is sound, but sound is most certainly not necessarily all there is about music, since music can also be said to exist with no sound present—the phenomenon of the 'earworm' is sufficient proof of this. This then raises the question what the role of sound is in music. Herein, we adopt the view of Wiggins (2009; 2010), which is in turn based on Milton Babbitt's work (1965). According to Wiggins, music can reside in three different domains (see Figure 1.1): the acoustic (or physical), the auditory (or perceived/internalised) or the graphemic (or stored/notated). Music as sound belongs to the acoustic domain, which embraces all the physical properties of sound. The graphemic domain can be viewed as an unlimited collective memory that is able to store musical information: this can be a musical score, but also a digital representation such as a CD or MP3 file. The auditory domain accounts for all musical phenomena that occur inside the human mind. Each of these domains captures a different aspect of music. All together these domains describe everything that we consider music, while music itself is something abstract and intangible that is constantly redefined by composers, musicians and listeners.

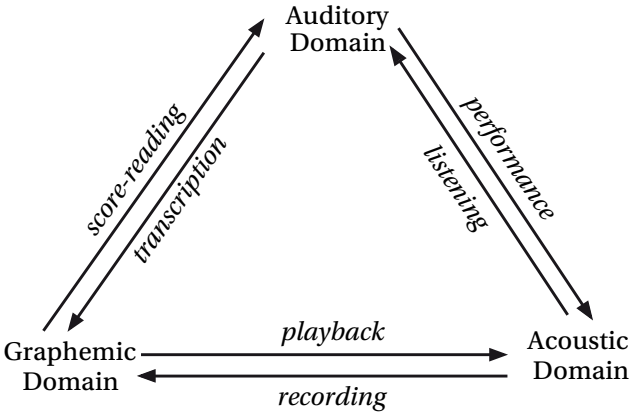


Figure 1.1: Babbitt's trinity of domains, with Wiggins' addition of transformations between them, quoted from Wiggins (2009).

Categorising musical phenomena in three different domains does not imply that all three domains are equally important. Music can exist without sound being present, since people can imagine music without listening to it, or even create it, like Beethoven when he was deaf. On the other hand, improvised music is often performed with little or no graphical information, and should ideally not be recorded, but experienced in a live setting. However, without human intervention there is no music. The fundamental source, but also the purpose of music can only be found in the human mind, without which music cannot exist. Therefore, a deeper understanding of what music is can only be gained by investigating the human mind.

From the point of view of an MIR researcher, the graphemic domain is particular interesting. Analogous to written language, the printing press, or photography, the graphemic domain emerged to compensate for one of our brains' major deficits: its lack of ability to precisely recall and/or reproduce something that happened in the past. This brings us back to the problem sketched at the beginning of this chapter of the immense amount of valuable, but often unorganised musical material that we want to unlock for a general audience. We believe that this can only be done in an automated fashion if the machine has a reasonable understanding of how this data is processed by the human mind. In turn, scientifically studying and formalising the cognition of music can only achieve this.

We are certainly not the first to call for a more music-cognitive inspired approach to MIR. Already at the first ISMIR in 2000, David Huron presented a paper entitled

'Perceptual and Cognitive Applications in Music Information Retrieval' (Huron 2000). Other MIR researches, like Aucouturier and Pachet (2004) and Celma and Serra (2006), recognised that data-only MIR methods suffered from a glass ceiling or a semantic gap. Also, scholars like Honing (2010) and Wiggins (2009; 2010) have been stressing the importance of music-cognitively oriented alternatives. We share this stance and believe that complementing low-level bottom-up with top-down approaches that start from what knowledge we already have about music, can have a positive effect on retrieval performance since they sidestep the issue of automatically assembling that knowledge in the first place.

Nonetheless, it is certainly not all doom and gloom in the field of MIR. There exist some successful MIR approaches that are based on musical knowledge. Some examples include: using perceptual models (Krumhansl 2001) to search for the musical key (Temperley 2001, Ch. 7, also see Chapter 6); improving harmonic similarity by performing automatic harmonic analyses (De Haas et al. 2009, 2011b, see also Chapter 5); or by consulting Lerdahl's (2001) Tonal Pitch Space (De Haas et al. 2008, see also Chapter 2); using musically enhanced alignment algorithms for searching similar folk songs (Van Kranenburg et al. 2009) and integrating this in a search engine for folk songs (Wiering et al. 2009b); enabling efficient retrieval of classical incipits by vantage point indexing and transportation distances (Typke 2007); making F0 estimation easier with a filter model based on human pitch perception (Klapuri 2005); using Gestalt principles for grouping musical units (Wiering et al. 2009a); or retrieving melodies on the basis of the Implication/Realization model (Narmour 1990; Grachten et al. 2004), to name a few.

Although we are convinced that MIR approaches grounded in musical knowledge have an advantage over merely data-oriented approaches, still some of the arguments posed in the previous section also hold for non-data-oriented approaches. However, we will argue below that these arguments do not have such severe consequences when MIR research does not rely on the musical data alone.

Ground-truth and data. Ground-truth data is essential to the evaluation of advances MIR and the lack of it will restrict MIR in progressing further. However, knowledge- or model-based approaches to MIR do not require ground-truth data for training, thus reducing the overall data need.

Danger of overfitting. Knowledge-based approaches are vulnerable to overfitting as well. After all, a specific parameter setting of a musical model that is optimal for a certain data set does not necessarily have to be optimal for other data sets. However, overfitting is interpretable and easier to control, because the parameters have a (musical) meaning and the MIR researcher can predict how the model will respond to new

data.

Curse of dimensionality. Prior knowledge about music can help an MIR researcher to reduce the dimensionality of the input space by transforming it into a lower dimensional feature set with more expressive power. To give a simple example, instead of representing a chord with notes in eight octaves a MIR researcher could assuming octave equivalence and choose to represent it with only twelve pitch classes. This reduces the dimensionality of the input vector and reduces the space of possible chords considerably.

Neglecting time. Music only exists in time. If a musical model disregards the temporal and sequential aspect of music, it fails to capture an essential part of the musical structure. Hence, it might be wise to reconsider the choice of model. Besides, there are plenty of musical models that do incorporate the notion of musical, e.g. models for segmenting music (Wiering et al. 2009a), musical expectation (Pearce and Wiggins 2006), timing and tempo (Honing and De Haas 2008), or (melodic) sequence alignment Van Kranenburg et al. (2009), to name a few.

Nothing to learn? Using an MIR approach based on cognitive models might not only be beneficial to the retrieval performance; it can also be used to evaluate the model at hand. When such an MIR system is empirically evaluated, also the model is evaluated by the experiment—albeit implicitly. The evaluation can provide new insights into the musical domain used and thereby contribute to the improvement of the musical model.

Not all information is in the data. This point has been extensively made above and needs no further explanation.

1.4 Harmony based MIR

As we stated in the beginning of this chapter, in this doctoral dissertation we investigate a very specific topic within MIR: the similarity, analysis and extraction of harmonic music information. Although the analysis of tonal harmony will be addressed thoroughly, and we will present a method for extracting harmonic information from musical audio, the main focus of this dissertation is on harmonic similarity. As we explained in the previous section, we strongly believe that MIR tasks are best accomplished using model-based approaches. However, before we expand on the MIR methods and harmony models of others and of our own making in the remainder of this dissertation, we will first present a brief introduction into Western tonal harmony and harmonic similarity.

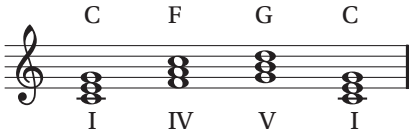


Figure 1.2: A very typical and frequently used chord progression in the key of C-major, often referred to as I-IV-V-I. Above the score the chord labels, representing the notes of the chords in the section of the score underneath the label, are printed. The Roman numbers below the score denote the interval between the chord root and the tonic of the key. We ignored voice-leading in this example for simplicity.

1.4.1 What is harmony?

The most basic element in music is a *tone*. A tone is a sound with a fixed frequency that can be described in a musical score with a *note*. All notes have a name, e.g. C, D, E, etc., and represent tones of specific frequencies. The distance between two notes is called an *interval* and is measured in semitones, which is the smallest interval in Western tonal music. Also intervals have names: minor second (1 semitone), second (2 semitones), minor third (3 semitones), etc., up to an octave (12 semitones). When two tones are an octave apart the highest tone will have exactly twice its frequency. These two tones are also perceived by the listeners as very similar, so similar even that all tones one or more octave apart have the same name. Hence, these tones are said to be in the same *pitch class*.

Harmony arises in music when two or more tones sound at the same time. These simultaneously sounding notes form chords, which can in turn be used to form chord sequences. A *chord* can be viewed as a group of tones that are often separated by intervals of roughly the same size. The most basic chord is the *triad* which consists of three pitch classes that are separated by two thirds. The two most important factors that characterise a chord are its structure, determined by the intervals between these notes, and the chord *root*. The root note is the note on which the chord is built. The root is often, but it does not necessarily have to be, the lowest sounding note. Figure 1.2 displays a frequently occurring chord sequence. The first chord is created by taking a C as root and subsequently a major third interval (E) and a minor third interval (G) are added, yielding a C-major chord. Above the score the names of the chords, which are based on the root of the chord, are printed. If the interval between the root of the chord and the third is a major third, the chord is called a *major chord*, if it is a minor third, the chord is called a *minor chord*.

The internal structure of the chord has a large influence on the *consonance* or *disson-*

ance of a chord: some combinations of simultaneous sounding notes are perceived to have a more tense sound than others. Another important factor that contributes to perceived tension of a chord is the relation between the chord and the *key* of the piece. The key of a piece of music is the tonal centre of the piece. It specifies the *tonic*, which is the most stable, and often the last, tone in that piece. Moreover, the key specifies the *scale*, which is set of pitches that will occur most frequently and that sound reasonably well together. A chord can be built up from pitches that are part of the scale or they can borrow notes from outside the scale, the latter being more dissonant. Especially the root note of a chord has a distinctive role because the interval of the chord root and the key largely determines the *harmonic function* of the chord. The three most important harmonic functions are the dominant (V), which builds up tension, a sub-dominant (IV), which prepares a dominant and the tonic (I) that releases tension. In Figure 1.2 a Roman number that represents the interval between the root of the chord and the key, often called *scale degree*, is printed underneath the score.

Obviously, this is a rather basic view on tonal harmony. For a thorough introduction to tonal harmony we refer to (Piston and DeVoto 1991). Harmony is considered a fundamental aspect of Western tonal music by musicians and music researchers. For centuries, the analysis of harmony has aided composers and performers in understanding the tonal structure of music. The harmonic structure of a piece alone can reveal song structure through repetitions, tension and release patterns, tonal ambiguities, modulations (i.e. local key changes), and musical style. Therefore, Western tonal harmony has become one of the most prominently investigated topics in music theory and can be considered a feature of music that is equally distinctive as rhythm or melody. Nevertheless, harmonic structure as a feature for music retrieval has received far less attention than melody and rhythm within the MIR field.

1.4.2 Harmonic similarity

Just like many of the other MIR tasks we touch upon earlier, harmonic similarity depends not only on the musical information, but also largely on the interpretation of this information by the human listener. Human listeners, musician and non-musician alike, have extensive culture-dependent knowledge about music that needs to be taken into account when modelling music similarity.

In this light we consider the harmonic similarity of two chord sequences to be the degree of agreement between structures of simultaneously sounding notes and the agreement between global as well as local relations between these structures in the two sequences as perceived by the human listener. By the agreement between structures of simultaneously sounding notes we denote the similarity that a listener perceives when

comparing two chords in isolation and without surrounding musical context. However, chords are rarely compared in isolation and the relations to the global context–the key of a piece–and the relations to the local context play a very important role in the perception of tonal harmony. The local relations can be considered the relations between functions of chords within a limited time frame, for instance the preparation of a chord with a dominant function by means of a sub-dominant. All these factors play a role in the perception of tonal harmony and should be shared by two compared pieces up to certain extent to be considered similar.

In the context of this view on harmonic similarity, music retrieval based on harmony sequences clearly offers various benefits. It allows for finding different versions of the same song even when melodies vary. This is often the case in cover songs or live performances, especially when these performances contain improvisations. Moreover, playing the same harmony with different melodies is an essential part of musical styles like jazz and blues. Moreover, also for finding song from a particular harmonic family, such as blues or rhythm changes, songs must be related harmonically. However, also variations over standard basses in baroque instrumental music can be harmonically closely related, e.g. chaconnes, or Bach's Goldberg variations.

1.5 Organisation of this dissertation

We have presented a brief overview of the field of MIR and argued that knowledge about music is necessary for improving the current state-of-the-art in MIR. We believe that the assumption that all information is in the data will hamper the development of high performance, usable and effective MIR systems and that in particular knowledge about music cognition and music theory can aid in overcoming the limitations that MIR is facing today. The methods, algorithms and models presented in this thesis underpin this argument and transport this idea to the area of tonal harmony by delivering novel, music theoretically sound, high performance, harmony based retrieval methods.

We begin in Chapter 2 by presenting a brief overview of the related work in polyphonic MIR. Subsequently, we present a geometric approach towards harmonic similarity: the Tonal Pitch Step Distance (TPSD). The similarity between the chords in a sequence is estimated by comparing the change of chordal distance to the tonic over time. For approximating the distance between the individual chords and the tonic we build on Lerdahl's Tonal Pitch Space (2001, TPS). The TPSD can be computed efficiently and matches human intuitions about harmonic similarity. To evaluate the retrieval performance, a large corpus of 5028 user-generated chord sequences is assembled. This corpus contains several chord sequences that describe the same piece in a different way.

Hence, these sequences can be considered harmonically related. The TPSD is evaluated in three different flavours and compared to a baseline string matching approach and an alignment approach. A surprising finding is that using only the triad of the chords in a sequence and discarding all other chord additions yields the best retrieval performance. Next, we compare the performance of the TPSD with two other measures of harmonic similarity. Additionally, we show in a case study how harmonic similarity measures, like the TPSD, can contribute to the musicological discussion about the relation between melody and harmony in melodically related Bach chorales.

In Chapter 3 we explore a first way of automatically harmonically analysing a sequence of chords and using the obtained functional annotations for similarity estimation. For this, we model tonal harmony as a formal grammar based on the ideas of Rohrmeier (2007). Given a sequence of symbolic chord labels we derive a tree structure that explains the function of a chord within its tonal context. Next, given two sequences of chords we define their similarity by analysing the properties of the largest labelled common embeddable subtree (LLCES). An algorithm is given for the calculation of the LLCES and six different similarity measures based on the LLCES are evaluated on a dataset of 72 chord sequences of jazz standards.

The proof-of-concept presented in Chapter 3 is interesting from a music theoretical point of view and the results on a small dataset of 72 chord sequences are good. Nevertheless, it also exposed some serious difficulties. How should we deal with ambiguous solutions, and how should we obtain a harmony analysis if the grammar rejects a chord sequence? In Chapter 4 we present HARMTRACE, a functional model of Western tonal harmony, that expands on the grammatical approach presented in Chapter 3, but solves the problems associated with context-free parsing by exploiting state-of-the-art functional programming techniques. In contrast to other formal approaches to tonal harmony, we present an implemented system that, given a sequence of chord labels, automatically derives the functional dependency relations between these chords. Our system is fast, easy to modify and maintain, and robust against noisy data. This is demonstrated by showing that the system can quickly parse all 5028 chord sequences of the corpus presented in Chapter 2. That the analyses make sense music theoretically is demonstrated by discussing some example analyses.

In Chapter 5 we extend the similarity estimation ideas in Chapter 3. We use the HARMTRACE harmony model to perform an automatic harmonic analysis and use the obtained harmonic annotations to improve harmonic similarity estimation. We present various common embeddable subtree based and local alignment based similarity measures and compare them to the currently best performing harmonic similarity measures. The results show that a HARMTRACE harmony model based alignment approach outperforms all other harmonic similarity measures, with a mean average precision of 0.722,

and that exploiting knowledge about the harmonic function of a chord improves retrieval performance.

Most similarity measures presented in this thesis have been using symbolic chord labels as their main representation for musical harmony. In Chapter 6 we present MPTrEE: a system that again employs the automatic harmony analyses of the HARMTRACE harmony model, but with a different purpose. We show how we can use the harmonic analyses to improve automatic chord transcription from musical audio and the output of the MPTrEE system can be directly used for harmony analysis and similarity estimation. Using the Vamp plugin architecture (Cannam et al. 2010), spectral and pulse features are extracted from the raw audio signal. Subsequently, given a dictionary of expected chord structures, we estimate the probabilities of the chord candidates matching a segment. If the spectrum clearly favours a particular chord, this chord is chosen to represent that segment. However, in case there is a lot of uncertainty in the data and multiple chords match the spectrum well, we let the harmony model decide which of the chord candidates fits the segment best from a music theoretical point of view. We demonstrate that a model of tonal harmony yields a significant chord transcription improvement on a corpus of 179 Beatles songs.

In the last chapter of this thesis we will discuss the implications of the most important conclusions of this thesis. At that point we hope to have convinced the reader about the potential of music theoretical and music cognitive models and their value for MIR.

1.6 Contributions of this dissertation

This doctoral dissertation covers a wide range of methods that can aid in the organisation of harmonic music information. We present three new approaches to harmonic similarity: a geometric, a local alignment, and a common embeddable subtree based approach. For each of these harmonic similarity solutions, the adjustable parameters are discussed and evaluated. For the evaluation a large new chord sequence corpus is assembled consisting of 5028 different chord sequences. We furthermore present a novel model of tonal harmony that can be used to automatically analyse harmony progressions. These harmonic annotations, which explain the role of a chord in its tonal context, can be used improve harmonic similarity estimation. Finally, we demonstrate how automatic chord transcription from musical audio can be improved by exploiting our model of tonal harmony.

1.7 Relevant publications

The work presented in this thesis is based on some articles that have been published before, or are still on under review. We list these articles below:

Chapter 1.

De Haas, W. B. and Wiering, F. (2010). Hooked on music information retrieval. *Empirical Musicology Review*, 5(4):176–185

Chapter 2.

De Haas, W. B., Veltkamp, R. C., and Wiering, F. (2008). Tonal pitch step distance: A similarity measure for chord progressions. In *Proceedings of the 9th International Society for Music Information Retrieval Conference (ISMIR)*, pages 51–56

De Haas, W. B., Robine, M., Hanna, P., Veltkamp, R., and Wiering, F. (2011c). Comparing approaches to the similarity of musical chord sequences. In Ystad, S., Aramaki, M., Kronland-Martinet, R., and Jensen, K., editors, *Exploring Music Contents*, volume 6684 of *Lecture Notes in Computer Science*, pages 242–258. Springer Berlin / Heidelberg

De Haas, W. B., Wiering, F., and Veltkamp, R. C. (under review 2011d). A geometrical distance measure for determining the similarity of musical harmony. *International Journal of Multimedia Information Retrieval (IJMIR)*

Chapter 3.

De Haas, W. B., Rohrmeier, M., Veltkamp, R. C., and Wiering, F. (2009). Modeling harmonic similarity using a generative grammar of tonal harmony. In *Proceedings of the 10th International Society for Music Information Retrieval Conference (ISMIR)*, pages 549–554

Chapter 4.

Magalhães, J. P. and De Haas, W. B. (2011). Functional modelling of musical harmony: an experience report. In *Proceeding of the 16th ACM SIGPLAN International Conference on Functional Programming (ICFP)*, pages 156–162, New York, NY, USA. ACM

De Haas, W. B., Magalhães, J. P., Veltkamp, R. C., and Wiering, F. (under review 2011a). HarmTrace: Automatic functional harmonic analysis. *Computer Music Journal*

Chapter 5.

De Haas, W. B., Magalhães, J. P., Wiering, F., and Veltkamp, R. C. (2011b). HarmTrace: Improving harmonic similarity estimation using functional harmony analysis. In *Proceedings of the 12th International Society for Music Information Retrieval Conference (ISMIR)*

Chapter 2

A geometrical distance measure for tonal harmony

CONTENT-BASED Music Information Retrieval (MIR) is a rapidly expanding area within multimedia research. On-line music portals like last.fm, iTunes, Pandora, Spotify and Amazon disclose millions of songs to millions of users around the world. Propelled by these ever-growing digital repositories of music, the demand for scalable and effective methods for providing music consumers with the music they wish to have access to, still increases at a steady rate. Generally, such methods aim to estimate the subset of pieces that is relevant to a specific music consumer. Within MIR the notion of *similarity* is therefore crucial: songs that are similar in one or more features to a given relevant song are likely to be relevant as well. In contrast to the majority of approaches to notation-based music retrieval that focus on the similarity of the *melody* of a song, this chapter presents a new method for retrieving music on the basis of its *harmony* structure.

Within MIR two main directions can be discerned: symbolic music retrieval and the retrieval of musical audio. The first direction of research stems from musicology and the library sciences and aims to develop methods that provide access to digitised musical scores. Here music similarity is determined by analysing the combination of symbolic entities, such as notes, rests, metre signs, etc., that are typically found in musical scores. Musical audio retrieval arose when the digitisation of audio recordings started to flourish and the need for different methods to maintain and unlock digital music collections emerged. Audio based MIR methods extract features from the audio signal and use these features for estimating whether two pieces of music are musically related.

Typical audio features, like chroma features (Wakefield 1999), Harmonic Pitch Class Profiles (HPCP; Gómez 2006), or Mel-Frequency Cepstral Coefficients (MFCCs; Logan 2000),can be used to directly estimate the similarity between two audio signals

(Müller et al. 2005) or for finding cover-songs (Serrà et al. 2008) However, these features do not directly translate to the notes, beats, voices and instruments that are used in the symbolic domain. Of course, much depends on the application or task at hand, but if the task requires judging the musical content of a musical audio, translating the audio features into notes, beats and voices and use such a high level representation is preferred. After all, how would one be able to estimate the musical function of a song segment and its relation to other segments within that piece or within another piece without having information about the tonal content? Unfortunately, current automatic polyphonic music transcription systems have not matured enough for their output to be usable for determining music similarity. Hence, in this chapter we focus on a symbolic musical representation that can be transcribed reasonably well from the audio signal using current technology: chord sequences. In other words, for applying our method to audio we assume a preprocessing step is made with one of the available chord transcription methods (See Section 2.1.2 and also Chapter 6).

Contribution. In this chapter we present a novel similarity measure for chord sequences. The similarity measure is based on a cognitive model of tonality and models the change of chordal distance to the tonic over time. The proposed measure can be computed efficiently and matches human intuitions about harmonic similarity. The retrieval performance is examined in an experiment on 5028 human-generated chord sequences, in which we compare it to two other harmonic distance functions. We furthermore show in a case study how the proposed measure can contribute to the musicological discussion about the relation between melody and harmony in melodically similar Bach chorales. The work presented here extends and integrates earlier harmony similarity work in (De Haas et al. 2008, 2011c).

We will discuss related work on harmonic similarity and the research from music theory and music cognition that is relevant for our similarity measure in Section 2.1.1. Next, we will present the Tonal Pitch Step distance in Section 2.2. In Section 2.3 we show how our distance measure performs in practice and we show that it can also contribute to musicological discussions in Section 2.4. But first, we will give a brief introduction on what actually constitutes tonal harmony and harmonic similarity.

2.1 Related work

MIR methods that focus on the harmonic information in the musical data are quite numerous. After all, a lot of music is polyphonic and limiting a retrieval system to melodic data only would restrict its application domain considerably. Most research seems to focus on complete polyphonic MIR systems, (e.g., Bello and Pickens 2005). By complete

systems we mean systems that do chord transcription, segmentation, matching and retrieval all at once. The number of papers that purely focus on the development and testing of harmonic similarity measures is much smaller. In the next section we will review other approaches to harmonic similarity, in Section 2.1.2 we will discuss the current state of automatic chord transcription, in Section 2.1.3, and in Section 2.1.4 we elaborate on the cognition of tonality and the cognitive model relevant to the similarity measure that will be presented in Section 2.2.

2.1.1 Harmonic similarity measures

All polyphonic similarity measures slice up a piece of music in segments that represent a single chord. Typical segment lengths range from the duration of a sixteenth note up to the duration of a couple of beats depending on the kind of musical data and the segmentation procedure.

An interesting symbolic MIR system based on the development of harmony over time is the one developed by Pickens and Crawford (2002). Instead of describing a musical segment as a single chord, they represent a musical segment as a 24 dimensional vector describing the 'fit' between the segment and every major and minor triad, using the euclidean distance in the for dimensional pitch space as found by (Krumhansl 2001) in her controlled listening experiments (see Section 2.1.3). Pickens and Crawford then use a Markov model to model the transition distributions between these vectors for every piece. Subsequently, these Markov models are ranked using the Kullback-Leibler divergence to obtain a retrieval result.

Other interesting work has been done by Paiement et al. (2005). They define a similarity measure for chords rather than for chord sequences. Their similarity measure is based on the sum of the perceived strengths of the harmonics of the pitch classes in a chord, resulting in a vector of twelve pitch classes for each musical segment. Paiement et al. subsequently define the distance between two chords as the euclidean distance between two of these vectors that correspond to the chords. Next, they use a graphical model to model the hierarchical dependencies within a chord progression. In this model they use their chord similarity measure for the calculation of the substitution probabilities between chords and not for estimating the similarity between sequences of chords.

Besides the similarity measure that we will elaborate on in this chapter and which was earlier introduced in (De Haas et al. 2008) there are two other methods that solely focus on the similarity of chord sequences: an alignment based approach to harmonic similarity (Hanna et al. 2009) and the grammatical parse tree matching method, which

will be introduced in Chapter 3 and further elaborated on in Chapter 5. The first two are quantitatively compared in an experiment in Section 2.3. The latter approaches will be compared with the TPSD and the approach of Hanna et al. in Section 5.5

The Chord Sequence Alignment System (CSAS; Hanna et al. 2009) is based on local alignment and computes similarity between two sequences of symbolic chord labels. By performing elementary operations the one chord sequence is transformed into the other chord sequence. The operations used to transform the sequences are deletion or insertion of a symbol, and substitution of a symbol by another. The most important part in adapting the alignment is how to incorporate musical knowledge and give these operations valid musical meaning. Hanna et al. experimented with various musical data representations and substitution functions and found a key relative representation to work well. For this representation they rendered the chord root as the difference in semitones between the chord root and the key; and substituting a major chord for a minor chord and vice versa yields a penalty. The total transformation from the one string into the other can be solved by dynamic programming in quadratic time. For a more elaborate description of the CSAS we refer to (Hanna et al. 2009) and (De Haas et al. 2011c).

2.1.2 Chord transcription

The application of harmony matching methods is extended by the extensive work on chord label extraction from musical audio and symbolic score data within the MIR community. Chord transcription algorithms extract chord labels from raw musical data and these labels can be matched using the distance measures presented in this chapter.

Currently there are several methods available that derive these descriptions from raw musical data. Recognising a chord in a musical segment is a difficult task: in case of audio data, the stream of musical data must be segmented, aligned to a grid of beats, the different voices of the different instruments have to be recognised, etc. Even if such information about the notes, beats, voices, bar lines, key signatures, etc. is available, as it is in the case of symbolic musical data, finding the right description of the musical chord is not trivial. The algorithm must determine which notes are unimportant passing notes and sometimes the right chord can only be determined by taking the surrounding harmonies into account. Nowadays, several algorithms can correctly segment and label approximately 84 percent of a symbolic dataset (see for a review: Temperley 2001). Within the audio domain hidden Markov Models are frequently used for chord label assignment, (e.g., Mauch 2010; Bello and Pickens 2005). Within the audio domain, the currently best performing methods have an accuracy just below 80 percent. Of course, these numbers depend on musical style and on the quality of the data. In

Chapter 6 we present a method for improving chord transcription from musical audio.

2.1.3 Cognitive models of tonality

As we explained in Chapter 1, only part of the information needed for sound similarity judgement can be found in the musical information. Musically trained as well as untrained listeners have extensive knowledge about music (Deliège et al. 1996; Bigand 2003) and without this knowledge it might not be possible to grasp the deeper musical meaning that underlies the surface structure. We strongly believe that music should always be analysed within a broader music cognitive and music theoretical framework, and that systems without such additional musical knowledge are incapable of capturing a large number of important musical features (De Haas and Wiering 2010).

Of particular interest for the current research are the experiments of Carol Krumhansl (2001). Krumhansl is probably best known for her probe-tone experiments in which subjects rated the stability of a tone, after hearing a preceding short musical passage. Non surprisingly, the tonic was rated most stable, followed by the fifth, third, the remaining tones of the scale, and finally the non-scale tones. Also, Krumhansl did a similar experiment with chords: instead of judging the stability of a tone listeners had to judge the stability of all twelve major, minor and diminished triads[7]. The results show a hierarchical ordering of harmonic functions that is generally consistent with music-theoretical predictions: the tonic (I) was the most stable chord, followed by the sub-dominant (IV) and dominant (V) etc. For a more detailed overview we refer to (Krumhansl 2001, 2004).

These findings can very well be exploited in tonal similarity estimation. Therefore, we base our distance function on a model that not only captures the result found by Krumhansl nicely, but is also solidly rooted in music theory: the Tonal Pitch Space model.

2.1.4 Tonal pitch space

The Tonal Pitch Space model (TPS; Lerdahl 2001) builds on the seminal ideas in the *Generative Theory of Tonal Music* (Lerdahl and Jackendoff 1996) and is designed to make music theoretical and music cognitive intuitions about tonal organisation explicit. Hence, it allows to predict the proximities between chords that correspond very well to the findings of Krumhansl and Kessler (1982). Although the TPS can be used to calculate distances between chords in different keys, it is more suitable for calculating

[7] A diminished triad is a minor chord with a diminished fifth interval.

(a)	octave (root) level:	0											(0)
(b)	fifths level:	0							7				(0)
(c)	triadic (chord) level:	0				4			7				(0)
(d)	diatonic level:	0		2		4	5		7		9	11	(0)
(e)	chromatic level:	0	1	2	3	4	5	6	7	8	9	10	11	(0)
		C	C♯	D	E♭	E	F	F♯	G	G♯	A	B♭	B	(C)

Table 2.1: The basic space of the tonic chord in the key of C major. The basic space of the tonic chord in the key of C major (C = 0, C♯ = 1, . . . , B = 11), from Lerdahl (2001).

distances within local harmonic contexts (Bigand and Parncutt 1999). Therefore the distance measure presented in the next Section only utilises the parts of TPS needed for calculating the chordal distances within a given key. The TPS is an elaborate model of which we present an overview here, but additional information can be found in (Lerdahl 2001, pages 47 to 59).

The TPS model is a scoring mechanism that takes into account the perceptual importance of the different notes in a chord. The basis of the model is the *basic space* (see Table 2.1), which resembles a tonal hierarchy of an arbitrary key. In Table 2.1 the basic space is set to C major. Displayed horizontally are all twelve pitch classes, starting with 0 as C. The basic space comprises five hierarchical levels (a-e) consisting of pitch class subsets ordered from stable to unstable. The first and most stable level (a) is the root level, containing only the root of a chord. The next level (b) adds the fifth of a chord. The third level (c) is the triadic level containing all other pitch classes that are present in a chord. The fourth level (d) is the diatonic level consisting of all pitch classes of the diatonic scale of the current key. The last and least stable level (e) is the chromatic level containing all pitch classes. Chords are represented at level a-c and because the basic space is hierarchical, pitch classes present at a certain level will also be present at subsequent levels. The more levels a pitch class is contained in, the more stable the pitch class is and the more consonant this pitch class is perceived by the human listener within the current key. For the C chord in Table 2.1 the root note (C) is the most stable note, followed by the fifth (G) and the third (E). It is no coincidence that the basic space strongly resembles Krumhansl's (2001) probe-tone data.

Table 2.2 shows how a Dm chord can be represented in the basic space of (the key of) C major. Now, we can use the basic space to calculate distances between chords by transforming the basic space of a certain chord into the basic space of another chord. In order to calculate the distance between chords, the basic space must first be set to the tonal context–the key–in which the two chords are compared. This is done by shifting pitch classes in the diatonic level (d) in such manner that they match the pitch classes of the scale of the desired key. The distance between two chords depends on two

		2									
		2							9		
		2			5				9		
0		2		4	5		7		9		11
0	1	2	3	4	5	6	7	8	9	10	11
C	C♯	D	E♭	E	F	F♯	G	G♯	A	B♭	B

Table 2.2: A Dm chord represented in the basic space of C major. Level d is set to the diatonic scale of C major and the levels a-c represent the Dm chord, where the fifth is more stable than the third and the root more stable than the fifth.

factors: the number of diatonic fifth intervals between the roots of the two compared chords and the number of shared pitch classes between the two chords. These two factors are captured in two rules: the Chord distance rule and the Circle-of-fifths rule (from Lerdahl 2001):

> CHORD DISTANCE RULE: $d(x,y) = j+k$, where $d(x,y)$ is the distance between chord x and chord y. j is the minimal number of applications of the Circle-of-fifths rule in one direction needed to shift x into y. k is the number of distinctive pitch classes in the levels (a-d) within the basic space of y compared to those in the basic space of x. A pitch class is distinctive if it is present in the basic space of y but not in the basic space of x.

> CIRCLE-OF-FIFTHS RULE: move the levels (a-c) four steps to the right or four steps to the left (modulo 7) on level d. If the chord root is non-diatonic j receives the maximum penalty of 3.

The Circle-of-fifths rule makes sense music theoretically because the motion of fifths can be found in cadences throughout the whole of Western tonal music and is pervasive at all levels of tonal organisation (Piston and DeVoto 1991). The TPS distance accounts for differences in weight between the root, fifth and third pitch classes by counting the distinctive pitch classes of the new chord at all levels. Two examples of calculation are given in Table 2.3. Example 2.3a displays the calculation of the distance between a C chord and a Dm chord in the key of C major. It shows the Dm basic space that has no pitch classes in common with the C major basic space (see Table 2.1). Therefore, all six underlined pitch classes at the levels a-c are distinct pitch classes. Furthermore, a shift from C to D requires two applications of the circle-of-fifth rule, which yields a total distance of 8. In example 2.3b one pitch class (G) is shared between the C major basic space and the G^7 basic space. With one application of the circle-of-fifth rule the total chord distance becomes 6.

		2							9					2					7					
		2							9					2			5		7				11
		2			5				9					2		4	5		7		9		11
0		2		4	5		7		9		11		0		2		4	5	6	7	8	9	10	11
0	1	2	3	4	5	6	7	8	9	10	11		0	1	2	3	4	5	6	7	8	9	10	11
C	C♯	D	E♭	E	F	F♯	G	G♯	A	B♭	B		C	C♯	D	E♭	E	F	F♯	G	G♯	A	B♭	B
(a)													(b)											

Table 2.3: Example 2.3a shows the distance between a C chord and a Dm chord, d(C,Dm) = 8. Example 2.3b shows the distance between C chord and G^7 chord both in the context of a C major key, d(C,G^7) = 6. The distinct pitch classes are underlined.

				4							11				2									
				4							11				2					9				
				4			7				11				2		5			9				
0		2		4	5		7		9		11			1	2		4	5	6	7		9		11
0	1	2	3	4	5	6	7	8	9	10	11		0	1	2	3	4	5	6	7	8	9	10	11
C	C♯	D	E♭	E	F	F♯	G	G♯	A	B♭	B		C	C♯	D	E♭	E	F	F♯	G	G♯	A	B♭	B
(a)													(b)											

Table 2.4: Example 2.4a shows the chord distance calculation for a G and and a Em chord in the context of C major, d(C,Em) = 7. Example 2.4b shows the calculation of the distance between D and a Dm chord in the context of a D major key, d(D,Dm) = 2. The distinct pitch classes are underlined.

Two additional examples are given in Table 2.4. Example 2.4a shows the calculation of the distance between a G and an Em chord in the key of C major. The G chord and the Em chord have four distinct pitch classes (the only have two pitch classes in common) and three applications of the circle-of-fifths rule are necessary to transform the G basic space into the Em basic space. Hence, the total distance is 7. Example 2.4b displays the distance between a D and a Dm in the context of a D major key. There is only one distinct, but non-diatonic, pitch class and no shift in root position yielding a distance of 2.

The original TPS model also supports changes of key by augmenting the chord distance rule that quantifies the number of fifth leaps[8] of the diatonic level (d) to match a new region, i.e. key (Lerdahl 2001, p. 60, termed *full version*). By shifting the diatonic level, the tonal context is changed and a modulation is established. Next, the model as described above is applied in the same manner, but with the diatonic level shifted to match the new key. A difficulty of the regional extension is that it features a rather liberal modulation policy, which allows for the derivation of multiple different modulation sequences. We do not use the regional chord distance rule for the distance measures here presented and we will explain why in the next section. Hence, explaining the regional distances rules is beyond the scope of this chapter and we refer the reader to (Lerdahl 2001) for an in depth coverage of the issues involved.

[8] shifts of 7 steps on level the chromatic level (e)

2.2 Tonal pitch step distance

On the basis of the TPS chord distance rule, we define a distance function for chord sequences, named the Tonal Pitch Step Distance (TPSD). A low score indicates two very similar chord sequences and a high score indicates large harmonic differences between two sequences. The central idea behind the TPSD is to compare the change of chordal distance to the tonic over time. Hereby we deviate from the TPS model in to ways: first, only use the within region chord distance rule and discard the regional shifts; second, we apply the chord distance rule not to subsequent chords, but to each chord in the song and the tonic triad of the global key of the song.

The reason for not using the regional chord distance rule is computational one: incorporating the regional level entails a combinatorial explosion of possible modulations. Lerdahl suggests to prefer sequences of modulations that minimize the total distance. However, (Noll and Garbers 2004, Section 5.5) demonstrate that even for a short excerpt of *Wagner's Parsifal* there exist already thousands of minimal paths. Hence, from a computational point of view it is costly to compute all possible modulations and it is unclear how one should select the right path among all shortest paths.

The choice for calculating the TPS chord distance between each chord of the song and the tonic triad of the key of the song, was a representational one: if the distance function is based on comparing subsequent chords, the chord distance depends on the exact progression by which that chord was reached. This is undesirable because very similar but not identical chord sequences can then produce radically different scores.

When we plot the chordal distance to the tonic over the time a step function appears. The height of a step is determined by the chordal distance between the chord and key and the length is determined by the duration of the chord in beats (which is assumed to be available in the chord sequence data). Next, he difference between two chord sequences can then be defined as the minimal area between the two step functions f and g over all possible horizontal shifts t of f over g (see Figure 2.1). These shifts are cyclic. To prevent longer sequences from yielding higher scores, the score is normalised by dividing it by the length of the shortest step function. Trivially, the TSPD can handle step functions of different length since the area between non-overlapping parts is always zero. Note that because the step functions represent the tonal distance to the tonic, this representation is a key-relative and information about the global key is required.

The calculation of the area between f and g is straightforward. It can be calculated by summing all rectangular strips between f and g, and trivially takes $O(n+m)$ time

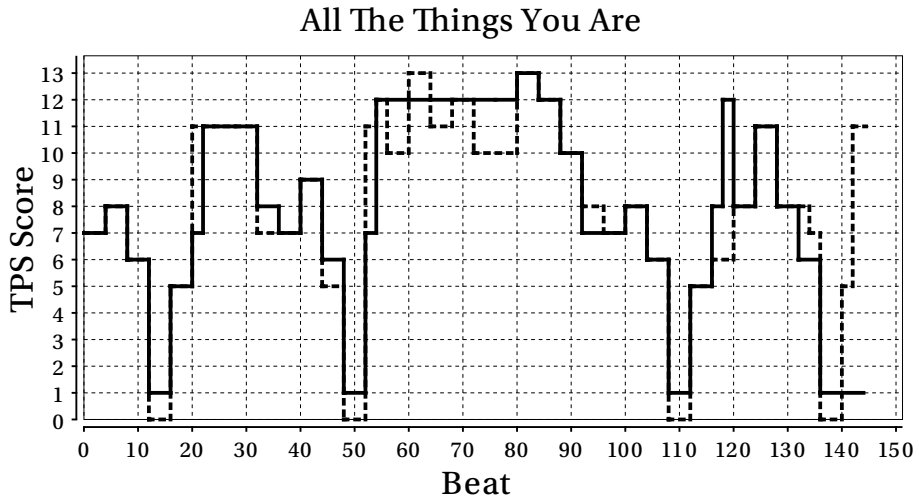


Figure 2.1: A plot demonstrating the comparison of two similar versions of *all the things you are* using the TPSD. The total area between the two step functions, normalised by the duration of the shortest song, represents the distance between both songs. A minimal area is obtained by shifting one of the step functions cyclically.

where n and m are the number of chords in f and g, respectively. An important observation is that if f is shifted along g, a minimum is always obtained when two vertical edges coincide. Consequently, only the shifts of t where two edges coincide have to be considered, yielding $O(nm)$ shifts and a total running time of $O(nm(n+m))$.

This upper bound can be improved. Arkin et al. (1991) developed an algorithm that minimised the area between two step functions by shifting it horizontally as well as vertically in $O(nm \log nm)$ time. The upper bound of their algorithm is dominated by a sorting routine. We adapted the algorithm of Arkin et al. in two ways for our own method: we shift only in the horizontal direction and since we deal with discrete time steps we can sort in linear time using counting sort (Cormen et al. 2001). Hence, we achieve an upper bound of $O(nm)$.

2.2.1 Metrical properties of the TPSD

For retrieval and especially indexing purposes it has several benefits for a distance measure to be a metric. The TPSD would be a metric if the following four properties held, where $d(x,y)$ denotes the TPSD distance measure for all possible chord sequences x and y:

									9															
				4					9									7	8					
0				4					9			0	1	2	3	4	5	6	7	8	9	10	11
0		2		4	5		7		9		11	0	1	2	3	4	5	6	7	8	9	10	11
0	1	2	3	4	5	6	7	8	9	10	11	0	1	2	3	4	5	6	7	8	9	10	11
C	C♯	D	E♭	E	F	F♯	G	G♯	A	B♭	B	C	C♯	D	E♭	E	F	F♯	G	G♯	A	B♭	B
(a)												(b)											

Table 2.5: An example of the minimal TPS chord distance and the maximal TPS chord distance. In example (a) two Am chords are compared yielding a distance of 0. In example (b) a C chord is compared to a C♯ chord with all possible additions resulting in a distance of 20. The distinct pitch classes are underlined. Note that pitch classes present a certain level are also present at subsequent levels.

1. *non-negativity:* $d(x,y) \geqslant 0$ for all x and y.
2. *identity of indiscernibles:* $d(x,y) = 0$ if and only if $x = y$.
3. *symmetry:* $d(x,y) = d(y,x)$ for all x and y.
4. *triangle inequality:* $d(x,z) \leqslant d(x,y) + d(y,z)$ for all x, y and z.

Before we look at the TPSD it is good to observe that the TPS model has a minimum and a maximum (see Table 2.5). The minimal TPS distance can obviously be obtained by calculating the distance between two identical chords. In that case there is no need to shift the root and there are no uncommon pitch classes yielding a distance of 0. The maximum TPS distance can be obtained, for instance, by calculating the distance between a C major chord and C♯ chord containing all twelve pitch classes. The Circle-of-fifths rule is applied three times and the number of distinct pitch classes in the C♯ basic space is 17. Hence, the total score is 20.

The TPSD is clearly non-negative since the length of the compared pieces, a and b, will always be $a \geqslant 0$ and $b \geqslant 0$, the area between the two step functions and hence the TPSD will always be $d(x,y) \geqslant 0$. The TPSD is symmetrical: when we calculate $d(x,y)$ and $d(y,x)$ for two pieces x and y, the shortest of the two step functions is fixed and the other step function is shifted to minimise the area between the two, hence the calculation of $d(x,y)$ and $d(y,x)$ is identical. However, the TPSD does not satisfy the identity of indiscernibles property because more then one chord sequence can lead to the same step function, e.g. C G C and C F C in the key of C major, all with equal durations. The TPS distance between C and G and C and F is both 5, yielding two identical step functions and a distance of 0 between these two chord sequences. The TPSD also does not satisfy the triangle inequality. Consider three chord sequences, x, y and z, where x and z are two different chord sequences that share one particular subsequence y. In this particular case the distances $d(x,y)$ and $d(y,z)$ are both zero,

but the distance $d(x,z) \geqslant 0$ because x and y are different sequences. Hence, for these three chord sequences $d(x,z) \leqslant d(x,y) + d(y,z)$ does not hold.

2.3 Experiment

The retrieval capabilities of the TPSD were analysed and compared to the CSAS in an experiment in which we tried to retrieve similar but not identical songs on the basis of similarity values of the tested distance functions. The TPSD distance between every sequence was calculated and for every song a ranking was constructed by sorting the other songs on the basis of their TPSD. Next, these rankings were analysed. To place the performance of these distance functions and the difficulty of the task in perspective, the performance of the TPSD was compared with an algorithm we call BASELINE. To measure the impact of the chord representation we compared three different flavours of both the TPSD as well as the CSAS: in a first task only the root note of the chord was available to the algorithms; in a second task we presented the root note and the triad of the chord (major, minor, augmented and diminished); and in a third task we presented the full chord with all extensions as they are found in the data.

For the triad and full chord tasks we used the TPSD as described in the previous section. We will denoted these variants of the TPSD with TPSDTRIAD and TPSDFULL, respectively. For the tasks where only the chord root was available we used a different step function representation. In these tasks the interval between the chord root and the root note of the key defined the step height and the duration of the chord again defined the step length. This matching method is very similar to the melody matching approach by Aloupis et al. (2004). Note that the latter was never tested in practice.

From the CSAS also different variants were evaluated. The first variant, CSASROOT, was evaluated in the root only task a +2 added to the total alignment score if the root note matched and −2 otherwise. In the chord triad task a the same procedure was followed: if the triad matched, a +2 was added and −2 if the triads did not match; this CSAS variant is named CSASTRIAD. In the full chord task Lerdahl's TPS model was used as a substitution function, this variant is denoted with CSASFULL.

The BASELINE similarity measure used the edit distance (Levenshtein 1966) between the two chord sequences represented as a string to quantify the similarity between the two chord sequences. However, one might consider this an unfair comparison because the TPSD and CSAS have more information it can exploit than the edit distance, namely information about the key. To make the comparison more fair we transposed all songs to C before matching the strings.

|Fm7 . . .	|Bbm7 . . .	|Eb7 . . .	|AbMaj7 . . . |
|DbMaj7 . . .	|Dm7b5 . G7b9 .	|CMaj7 . . .	|CMaj7 . . . |
|Cm7 . . .	|Fm7 . . .	|Bb7 . . .	|Eb7 . . . |
|AbMaj7 . . .	|Am7b5 . D7b9 .	|GMaj7 . . .	|GMaj7 . . . |
|A7 . . .	|D7 . . .	|GMaj7 . . .	|GMaj7 . . . |
|Gbm7 . . .	|B7 . . .	|EMaj7 . . .	|C+ . . . |
|Fm7 . . .	|Bbm7 . . .	|Eb7 . . .	|AbMaj7 . . . |
|DbMaj7 . . .	|Dbm7 . Gb7 .	|Cm7 . . .	|Bdim . . . |
|Bbm7 . . .	|Eb7 . . .	|AbMaj7 . . .	|. . . . |

Table 2.6: The chord sequence of the song *All the things you are*. A dot represents a beat, a bar represents a bar line, and the chord labels are presented as written in the Band-in-a-Box file.

2.3.1 A chord sequence corpus

For this experiment a large corpus of musical chord sequences is assembled. The chord sequence corpus consists of 5,028 unique human-generated Band-in-a-Box files that are collected from the Internet. Band-in-a-Box is a commercial software package (Gannon 1990) that is used to generate musical accompaniment based on a lead sheet. A Band-in-a-Box file stores a sequence of chords and a certain style, whereupon the program synthesises and plays a MIDI-based accompaniment. A Band-in-a-Box file therefore contains a sequence of chords, a melody, a style description, a key description, and some information about the form of the piece, i.e. the number of repetitions, intro, outro etc. For extracting the chord label information from the Band-in-a-Box files we have extended software developed by Simon Dixon and Matthias Mauch (2007). An example of a chord sequence as found in a Band-in-a-Box file describing the chord sequence of *All the Things You Are* is given in Table 2.6.

All songs of the chord sequence corpus were collected from various Internet sources. These songs were labelled and automatically checked for having a unique chord sequence. All chord sequences describe complete songs and songs with fewer than 3 chords or shorter than 16 beats were removed from the corpus. The titles of the songs, which function as a ground-truth, as well as the correctness of the key assignments, were checked and corrected manually. The style of the songs is mainly jazz, latin and pop.

Within the collection, 1775 songs contain two or more similar versions, forming 691 classes of songs. Within a song class, songs have the same title and share a similar melody, but may differ in a number of ways. They may, for instance, differ in key and form, they may differ in the number of repetitions, or have a special introduction or ending. The richness of the chords descriptions may also diverge, i.e. a $C^{7\flat9\flat13}$ may be

Class Size	Frequency	Percent
1	3,253	82.50
2	452	11.46
3	137	3.47
4	67	1.70
5	25	.63
6	7	.18
7	1	.03
8	1	.03
10	1	.03
Total	5028	100

Table 2.7: The distribution of the song class sizes in the chord sequence corpus

written instead of a C^7, and common substitutions frequently occur. Examples of the latter are relative substitution, i.e. Am instead of C, or tritone substitution, e.g. $F\sharp^7$ instead of C^7. Having multiple chord sequences describing the same song allows for setting up a *cover-song* finding experiment. The title of the song is used as ground-truth and the retrieval challenge is to find the other chord sequences representing the same song.

The distribution of the song class sizes is displayed in Table 2.7 and gives an impression of the difficulty of the retrieval task. Generally, Table 2.7 shows that the song classes are relatively small and that for the majority of the queries there is only one relevant document to be found. It furthermore shows that 82.5% of the songs is in the corpus for distraction only. The chord sequence corpus is available to the research community on request.

2.3.2 Results

Figure 2.2 shows the interpolated average precision calculated probed at 11 different recall levels, calculated as proposed in (Manning et al. 2008). In all evaluations the queries were excluded from the analysed rankings. The graph shows clearly that the overall retrieval performance of all algorithms can be considered good, and that the CSAS outperforms the TPSD, and both the TPSD and the CSAS outperform the BASELINE edit distance.

We also calculated the Mean Average Precision (MAP). The MAP is a single-figure measure, which measures the precision at all recall levels and approximates the area

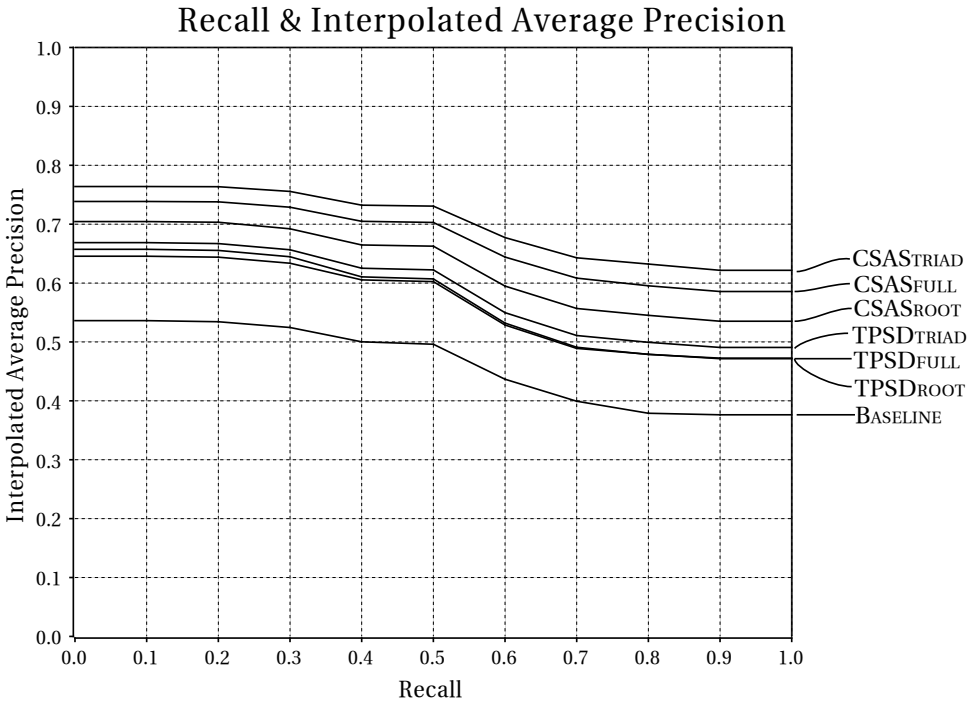


Figure 2.2: The interpolated average precision measured at 11 recall levels of the BASELINE, CSAS and TPSD. The latter two are evaluated in three tasks in which the amount of chord information is varied.

under the (uninterpolated) precision recall graph (Manning et al. 2008). Having a single measure of retrieval quality makes it easier to evaluate the significance of the differences between results. We tested whether the differences in MAP were significant by performing a non-parametric Friedman test, with a significance level of $\alpha = .01$. We chose the Friedman test because the underlying distribution of the data is unknown and in contrast to an ANOVA the Friedman does not assume a specific distribution of variance. There were significant differences between the runs, $\chi^2(6, N = 1775) = 896$, $p < .0001$. To determine which of the pairs of measurements differed significantly we conducted a post hoc Tukey HSD test[9]. Opposed to a T-test the Tukey HSD test can be safely used for comparing multiple means (Downie 2008). Table 2.9 displays the pairwise differences.

Most differences can be considered statistically significant. Only the differences

[9] All statistical tests were performed in Matlab 2009b.

	CSASTRIAD	CSASFULL	CSASROOT	TPSDTRIAD	TPSDFULL	TPSDROOT	BASELINE
MAP	0.696	0.666	0.622	0.580	0.565	0.559	0.459

Table 2.8: The mean average precision of the rankings based on the compared similarity measures.

	CSASFULL	CSASROOT	TPSDTRIAD	TPSDFULL	TPSDROOT	BASELINE
CSASTRIAD	+	+	+	+	+	+
CSASFULL		+	+	+	+	+
CSASROOT			−	−	+	+
TPSDTRIAD				+	+	+
TPSDFULL					−	+
TPSDROOT						+

Table 2.9: The pairwise statistical significance between all similarity measures. A + denotes a statistically significant difference and a − denotes a non-significant difference. The + and − signs were derived by pairwise comparison of the confidence intervals.

between CSASROOT and TPSDTRIAD; between CSASROOT and TPSDFULL; and between TPSDFULL and TPSDROOT were not statistically significant. Hence, we can conclude both the CSAS and the TPSD outperform the BASELINE method and that, irrespective of the kind of chord representation, the CSAS outperforms the TPSD. This does not mean that the chord representation does not have an effect. It is surprising to observe that the triad representation significantly outperforms the other representations for both the CSAS as well as the TPSD. It is furthermore interesting to see that using only the root of the chord already yields very good results, which in some cases is not even statistically different from using the full chord specification. Apparently, discarding chord additions acts as a form of syntactical noise-reduction, since these additions, if they do not have a voice leading function, have a rather arbitrary character and can only add some harmonic spice.

2.4 Relating harmony and melody in Bach's chorales

In this Section we show how a symbolic chord transcription algorithm can be combined with the TPSD and demonstrate how the TPSD can aid in answering musicological questions. More specifically, we will investigate whether melodically related chorale settings by J.S. Bach (1685-1750) are also harmonically related. Doing analyses of this kind by hand is very time consuming, especially when the corpus involved has a substantial size. Moreover, the question whether two pieces are harmonically related can hardly be answered with a simple yes or no. Pieces are harmonically similar up to a certain degree; forcing a binary judgement requires placing a threshold that is not

trivial to choose and maybe not even meaningful from a musical point of view. However, for a well-trained musicologist determining whether two melodies stem from the same tune family is a relatively simple task.

Chorales are congregational hymns of the German Protestant church service (Marshall and Leaver 2012). Bach is particularly famous for the imaginative ways in which he integrated these melodies into his compositions. Within these chorale-based compositions, the so-called *Bach chorales* form a subset consisting of relatively simple four-voice settings of chorale melodies in a harmony-oriented style often described as 'Cantionalsatz' or 'stylus simplex'. Bach wrote most of these chorales as movements of large-scale works (cantatas, passions) when he was employed as a church musician in Weimar (1708-1717) and Leipzig (1723-1750) (Wolff et al. 2011). A corpus of Bach chorales consisting of 371 items was posthumously published by C.P.E. Bach and J.P. Kirnberger in 1784-87, but some more have been identified since. This publication had a didactic purpose: the settings were printed as keyboard scores and texts were omitted. Consequently, over the last two centuries, the chorales have been widely studied as textbook examples of tonal harmony. Nevertheless, they generally provide very sensitive settings of specific texts rather than stereotyped models and, despite their apparent homogeneity, there is quite some stylistic variation and evidence of development over time. Yet one can claim that Bach's chorale harmonisations were constrained by the general rules of tonal harmony in force in the first half of the 18^{th} century and that the range of acceptable harmonisations of a given melody was limited.

We hypothesise that if two melodies are related, the harmonisations are also related and melodically similar pieces are also harmonically similar. To determine whether the melodies of two chorales are indeed related, we asked an expert musicologist to inspect the melodies that have the same title and to decide if these melodies belong to the same tune family. If they do, it should be possible to retrieve these settings by ranking them on the basis of their TPSD distance.

2.4.1 Experiment

To test whether the melodically related Bach Chorales were also harmonically related, we performed a retrieval experiment similar to the one in Section 2.3. We took 357 Bach Chorales and used the TPSD to determine how harmonically related these chorales were. Next, we used every chorale that belonged to a tune family, as specified by our musicological expert, as a query, yielding 219 queries, and created a ranking based on the TPSD. Subsequently we analysed the rankings with standard retrieval performance evaluation methods to determine whether the melodically related chorales could be found on the basis of their TPSD.

Tune Family Size	Frequency	Percent
1	136	68.34
2	24	12.06
3	17	8.54
4	10	5.03
5	5	2.51
6	3	1.51
7	4	2.01
11	1	0.50
Total	357	100

Table 2.10: Tune family distribution in the Bach chorales corpus

The chorales scores are freely available[10] in MIDI format (Loy 1985). But as explained in the previous sections, the TPSD takes chords as input, not MIDI notes. We therefore use David Temperley's Chord root tracker (Temperley 2001), which is part of the Melisma music analyser[11]. The chord root tracker does not produce a label for a segment of score data like we have seen in the rest of this chapter. It divides the piece into chord spans and it assigns a root label to each chord span. Thus, it does not produce a complete chord label, e.g. A♭m^9 but, this is not a problem, because the TPS model needs only to know which pitch class is the root and which one is the fifth. Once it is know which pitch class is the root, it is trivial to calculate which pitch class is the fifth. The remainder of the pitch classes in the chord is placed at level c of the basic space. The Melisma chord root tracker is a rule-based algorithm. It utilises a metrical analysis of the piece performed by the metre analyser, which is also part of the Melisma Music analyser, and uses a small number of music theoretically inspired preference rules to determine the chord root. We segmented the score such that each segment contained at least two simultaneously sounding notes. Manually annotating a small random sample yields a correctness of the root tracker of approximately 80%, which is in line with the 84% as claimed in (Temperley 2001).

The TPSD also requires to know the key of all chorales. The key information was generously offered by Martin Rohrmeier, who investigated the the distributions of the different chord transitions within the Chorales Corpus (Rohrmeier and Cross 2008). We selected the chorales of which both the MIDI data, a pdf score (for our musicological expert) and the key description was available. After preparation, which included

[10] See `http://www.jsbchorales.net` (accessed 18 January 2012) for more information.

[11] The source code of the Melisma Music Analyser is freely available at: `http://www.link.cs.cmu.edu/music-analysis/` (accessed 18 January 2012).

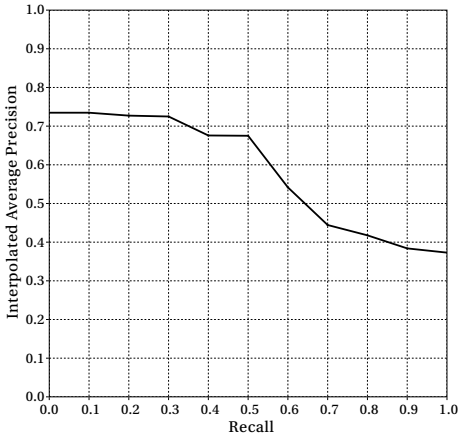


Figure 2.3: The average interpolated precision for ten different recall levels of the melodically related Bach chorales retrieved on the basis of their TPSD scores.

checking for chorale doublets, the corpus contained 357 pieces.

2.4.2 Results

We analyse the TPSD based rankings of Bach's chorales with an average interpolated precision versus recall plot, which is displayed in the graph in Figure 2.3. To place the results into context and have an idea of the structure of the corpus, we also printed the distribution of the sizes of the tune families in Table 2.10. The graph in Figure 2.3 shows clearly that a large section of the chorales that are based on the same melody can be found by analysing only their harmony patterns. In general we can conclude that in some melodically similar pieces can be found by looking at their harmony alone. This is supported by a recognition rate, i.e. the percentage of queries that have a melodically related chorale at rank one (excluding the query), of .71. However, a considerable amount of pieces cannot be retrieved on the basis of their TPSD: in 24 percent of the queries had the first related chorale is not within the first ten retrieved chorales.

This can have three reasons: the chorales are not harmonically related, the TPSD did not succeed in capturing the harmonic similarity well enough, or errors in the automatic chord transcription disturb the similarity measurement. We made a non-exhaustive analysis of the retrieval output in order to get a better idea of the issues at stake, focusing on the larger tune families. First, it appears that for some chorales the retrieval performance is very high. Perfect retrieval was attained for *Wo Gott, der Herr,*

nicht bei uns hält (5 items), *Was Gott tut das ist wolgetan* and *Wenn mein Stundlein* (both 4 items). Tune families with near-perfect retrieval include *Jesu meine Freude; Werde munter, mein Gemüte* (both 6 items, 2 false positives in total) and *Auf meinen lieben Gott* (5 items, 1 false positive in total). Retrieval is also very good for the largest group, *O Welt, ich muß dich lassen* (11 items). For each member all of the top-5 hits are from the same tune family, and for most members all other items are ranked within the top-20. Only one item has more than one relevant item ranked below 20 (BWV[12] 394).

Herzlich tut mich verlangen (7 items) presents a musically interesting situation: there seem to be two clusters, one of four and one of three items. Chorales from each of the two clusters match very well to one another, but chorales from the other cluster are consistently ranked low. From a melodic point of view, there are only a few unimportant differences. The harmony is very different for the two groups, though. The four-item cluster consists of settings in the major mode, with the chorale melody ending on the third of the final chord. The three-item cluster contains settings that are in the minor mode: the chorale melody ends on the root of the final chord, but this chord itself acts as a V in the rest of the piece. Generally, the larger tune families seem to consist of a cluster of very similar items and one or two items that fall outside the clusters. These 'outliers' generally rank the clustered items relatively low. There are some factors that may explain outliers:

Different metre. The default metre for chorale melodies is $\frac{4}{4}$. However, variants in $\frac{3}{4}$ exist for several chorales. In these the basic rhythmic pattern of two quarter notes is changed into a half note followed by a quarter note. This has three effects: the total length of the melody changes, some chords are extended because they follow the durations of the melody notes, and extra chords may be inserted on the second part of the half notes. All three factors lead to a high TPSD score when comparing chorale settings from the same tune family with different metres. Examples include *Wie nach einer Wasserquelle* (two outliers in $\frac{3}{4}$ metre) and *Nun lob, mein Seel, den Herren* (three versions in $\frac{3}{4}$, one, the outlier, in $\frac{4}{4}$ metre).

Phrase length. Individual phrases in a chorale melody typically end with a note with a fermata, which may or may not have indicated a prolongation of this note in performance. Sometimes however, fermatas are written out, replacing a quarter note by a dotted half note. Also, notes within the phrase are sometimes extended. Both of these situations create an asynchrony in the step function that contributes to a higher TPSD score. Both situations occur in the two versions of the melody *O Ewigkeit, du*

[12]The Bach-Werke-Verzeichnis (BWV) is a numbering system designed to order an identify the compositions by Johann Sebastian Bach. The works are numbered thematically, not chronologically and the prefix BWV, followed by the work's number has become a standard identifier for Bach's compositions.

Donnerwort, so that the two settings match each other particularly badly.

Additional instrumental parts. Some of the chorales have additional instrumental parts. If they are written in the same style as the vocal parts, this seems to present no particular problems. However, when they are different, this may lead to a higher TPSD score. An example of this is *Die Wollust dieser Welt* (4 settings, 1 outlier). The outlier has an instrumental bass moving in eighth notes, which leads many additional chord labels on weak beats. Since these labels are often dissonant chords, the TPSD score with 'normal' settings–which would have the second half of a more consonant chord at the corresponding place–increases.

Differences in polyphony. There are a number of settings that are much more polyphonic than most of the others. Some of these may actually be instrumental organ works written out in four voices. The rhythmic and melodic behaviour of the voices is very different. An example is *Christ lag in Todesbanden* (5 items, 2 outliers). Of the outliers, BWV 278 is particularly noticeable for its inner voices moving often in sixteenth notes and chromaticism. Here too a likely explanation is that extra, often dissonant chord labels are generated.

The last two points are related to a limitation of the TPSD, namely that all chords are considered equally important to the overall perception of harmonic similarity. In fact, chords have hierarchical relationships to each other, and in addition their contribution depends on the metric position of their onsets.

False negatives, items that get a high rank but belong to a different tune family, are informative as well. Sometimes these indeed appear to have an interesting relationship, as in the case of *Was mein Gott will*. Two settings of this melody also retrieve items with the melody *Wo Gott, der Herr, nicht bei uns hält*. It appears that the harmony of the first eight bars is very similar and that the melodies themselves also could be considered related. However, most of the false negatives are difficult to interpret. One reason is the cyclic shifting, which causes an arbitrary alignment between items that disrupts the phrase structure or may even lead to a match that includes a jump from the end of the piece to its beginning. Another reason is that different chords may have the same TPSD score, and that similar step functions may be generated by chord sequences that are musically quite different.

A different way of analysing false negatives is by looking into the average rank of each item over all queries. Ideally, the average rank should be normally distributed over all items in the collection, with a mean of half the collection size and a small standard deviation. Deviations from this ideal indicate that the similarity measure is sensitive to certain properties in the collection. In particular, items with a high average rank are

likely have certain properties that make them match to a large number of unrelated items. We studied the 15 pieces with the highest average rank and the 15 pieces with the lowest average rank and found clear patterns. The 15 pieces with the highest rank were all pieces in a minor key and the pieces with the lowest average rank were mainly major. Also, the pieces with a low average rank tend to be relatively long and the high-ranked ones tend to be relatively short. This indicates that the TPSD is susceptible to differences in length and key.

Nevertheless, we can conclude that a considerable number of pieces of the Bach chorales corpus that share the same melody could be shown to be also harmonically related.

2.5 Concluding remarks

We presented a new geometric distance measure that captures the harmonic distance between two sequences of musical harmony descriptions, named the Tonal Pitch Step Distance (TPSD). This distance is based on the changes of the distance between chord and key as given by using Lerdahl's Tonal Pitch Space model. This cognitive model correlates with empirical data from psychology and matches music-theoretical intuitions. A step function is used to represent the change of chordal distance to its tonic over time and the distance between two chord progressions is defined as the minimal area between two step functions. The TPSD is a distance measure that is simple to use, does not require a lot of parameter tuning, is key invariant, can be computed efficiently, and matches human intuitions about harmonic similarity.

The performance of the TPSD can still be considered good, especially if one considers the size of the test corpus used in the experiment and the relatively small class sizes (see Table 2.7). We compared the performance of the TPSD to the performance of the BASELINE string matching approach and a Chord Sequence Alignment System (CSAS). Both the TPSD and the CSAS significantly outperform the BASELINE string matching approach. In turn, the CSAS outperforms the TPSD significantly. Surprisingly, only using the root note of a chord gives good retrieval results. In case of the TPSD, the difference between using only the root is not even statistically different from using full chord specifications. Removing all chord additions and using only the triad significantly improves these results for both similarity measures. We furthermore demonstrated how the TPSD can contribute to the musicological discussions on melody and harmony in Bach's Chorales in a case study. In this case study we showed that a for a considerable number of Bach Chorales that share a melody also are harmonically related.

Nevertheless, there is still room for improvement. The TPSD cannot deal with large structural changes, e.g. adding repetitions, a bridge, etc. A prior analysis of the struc-

ture of the piece combined with partial matching could improve the retrieval performance. However, also when matching two identical sequences, inserting a chord in the middle of one of the two sequences will create a harmful asynchrony. Another important issue is that the TPSD treats all chords as equally important. This is musicologically not plausible. Considering the musical function in the local as well as global structure of the chord progression will improve retrieval results as we will see in Chapter 5.

The performance of the TPSD was only tested on symbolic data in this chapter. Nevertheless, the application TPSD is not limited to symbolic music and audio applications are currently investigated. Especially the recent developments in chord label extraction are very promising because the output of these methods could be matched directly with the systems here presented (e.g. Chapter 6). The good performance of the TPSD lead us to believe that also in other musical domains, such as audio retrieval systems will benefit from harmonic similarity based matching in the near future.

Chapter 3

Towards context-aware harmonic similarity

HARMONIC similarity is a relatively new research topic within Music Information Retrieval (MIR) that, by determining the similarity of the chord sequences, investigates searching for songs based on their harmony. Retrieval based on harmony offers obvious benefits: it allows for finding cover songs, especially when melodies vary; songs of a certain family, like blues or rhythm changes; or variations over standard basses in instrumental baroque music; to name a few. In Chapter 2 we gave an elaborate introduction into harmonic similarity and presented a geometrical approach to harmonic similarity. Like in this previous chapter, we assume that a sequence of symbolic chord labels is available and we aim to capture the similarity of two chord progressions in a single number. In this chapter we present a proof-of-concept of a different kind of harmonic similarity based on harmonic analysis. By implementing a model of tonal harmony as a context-free grammar, we can automatically analyse a chord progression and use the obtained harmonic annotations for similarity estimation.

As we elaborately argued in Chapter 1, we strongly believe that only part of the information needed for good similarity judgement can be found in the musical data. Musically trained as well as untrained listeners have extensive knowledge about music (Deliège et al. 1996; Bigand 2003) and one important task of a MIR researcher is to select or develop the appropriate music cognitive and music theoretical models that provide the knowledge needed for making good similarity judgements. In our view such models are necessary, and we believe systems without such additional musical knowledge are incapable of capturing a large number of important musical features.

The French-American composer Edgard Varèse once defined music as "organised sound" (Griffiths 2012). As we explained in Chapter 1 we believe that this organisation is not solely implicitly present in the music, but that it is also projected onto the musical information by the cognitive structures in our musical brain. Nevertheless, the organisational aspect is without a doubt an important and very prominent aspect of music,

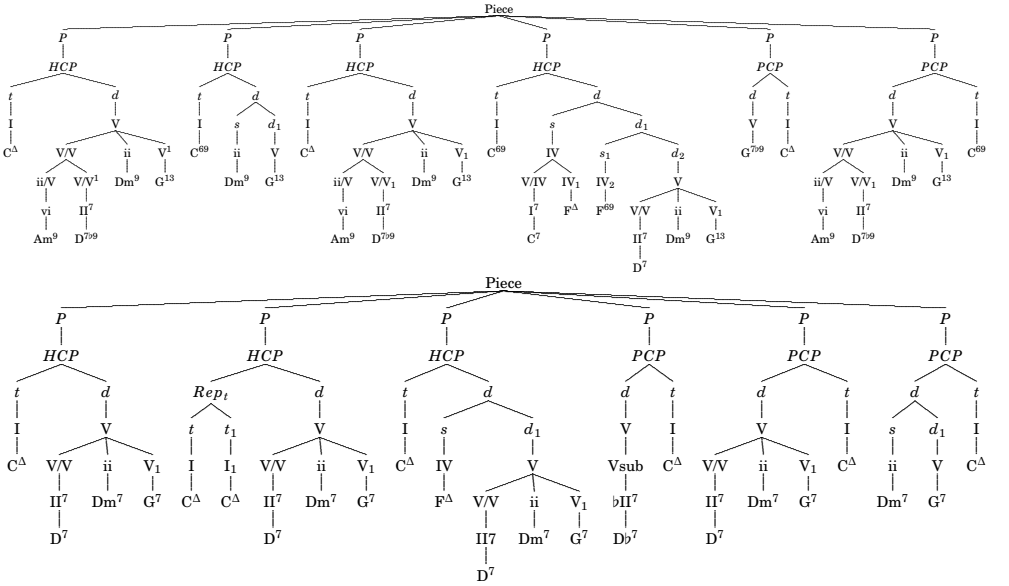


Figure 3.1: Two parse trees of different versions of the same jazz standard *Take the 'A' train*. The leaves of the tree represent the actual chords of the sequence.

and in this chapter we will argue that the organisation, at least from a harmonic point of view, should be a hierarchical one. As we will explain, this is a core assumption that underlies the models presented in this thesis. The hierarchical character of tonal harmony can be demonstrated by the phenomenon that the same chord can have different functions in different tonal contexts because these context dependent relationships between chords can only be explained in a hierarchical structure in which the same chord can fulfil different roles. Figure 3.1, displaying two versions of the jazz standard *Take the 'A' train*, illustrates this idea. Even though both sequences appear to be substantially different when compared element by element, an analysis of their harmonic dependencies reveals that both derive from a common harmonic pattern that is represented by the parse trees and fits human intuition.

Contribution. In the proof-of-concept presented in this chapter we perform a first attempt to model tonal harmony and exploit this model to improve harmonic similarity estimation. We present a fully functional remodelling of Rohrmeier's (2007) generative grammar of tonal harmony, which parses sequences of symbolic chord labels and returns parse trees like the ones in Figure 3.1. A parse tree represents an extensive harmonic analysis. It contains harmonic annotations at different levels: it distinguishes harmonic phrases, functional annotations (tonic, dominant, and subdominant), scale

degree transformations (e.g. tritone substitution) and the relations between these elements. This makes these parse trees very suitable for determining harmonic similarity because these different elements of a harmonic analysis allow for capturing harmonic similarity at different musical levels. We compare parse trees by finding and examining the combined labelled Largest Common Embeddable Subtree (LCES), for which we present a new algorithm. The LCES is the tree that can be included in both parse trees while maintaining the labels and the ancestor relations. Using the LCES we define a series of similarity measures for tonal harmony and compare them against a baseline string matching approach on 72 chord sequences.

In the following section, we will very briefly touch upon some of the most prominent formal theoretical approaches towards music and harmony theory. In Section 3.2 we present a remodelling of a formal grammar for tonal harmony and propose solutions for some of the typical problems of its application. Next, in Section 3.3 we present a new $O(\min(n,m)nm)$ time algorithm that calculates the LCES, where n and m are the sizes of the trees. Subsequently, the six LCES based distance measures defined in Section 3.3.3 are experimentally verified in Section 3.4. Finally, we discuss the limitations of the proof-of-concept presented in this chapter in Section 3.5.

3.1 Related work

Hierarchical models of music have a long history. In the late 19^{th} century Riemann (1893) first referred to the three main functional roles that chords have within a key. His grouping of a piece into keys, functions and scale degrees can be seen regarded as one of the first hierarchical orderings of tonal music. Almost 100 years later, one of the most important musical models of is the generative theory of tonal music (Lerdahl and Jackendoff 1996). Formalising Schenker's theory (1935), Lerdahl and Jackendoff developed an analysis framework for tonal music based on constraint-based preference rules. They structured Western tonal music by defining recursive hierarchical dependency relationships between musical elements and distinguish four kinds of structure: meter, grouping, time-span reduction, and prolongational reduction. Similarly, there is some theoretical evidence that tonal harmony is organised in a comparable, hierarchical way. Early attempts by Kostka and Payne (1984, ch. 13) and Baroni et al. (1983) suggest that harmony is organised in hierarchical layers. Later Steedman (1984, 1996) created a categorial grammar for explaining the twelve-bar blues progression. The articles just touched upon only constitute the most important contributions to the field of harmonic modelling, in Section 4.1 we discuss a number of additional models and for an elaborate review of all important approaches to formal harmony theory we refer to (Rohrmeier 2011).

Essential for the work presented in this chapter has been the work of Martin Rohrmeier (2007). Following among others Schenker (1935) and Lerdahl and Jackendoff (1996), Rohrmeier transports the ideas of hierarchical organisation of music to the area of tonal harmony. His model is based on the assumption that, within a sequence of harmonies, different chords have different degrees of stability and dependency, based on their position within the hierarchical structure. In a chord sequence several chords may be replaced, inserted or omitted in such a way that the harmonic structure remains intact, whereas the changes of structurally important anchor chords may result in a harmonically very different sequence. These dependency features and relationships resemble constituent structure and dependencies in linguistic syntax and can be modelled with a context-free grammar (Chomsky 1957).

3.2 A grammar for tonal harmony

In this section we present a remodelling of Rohrmeier's grammar without modulation and with limited local tonicization in order to reduce the complexity of the model. As we will see later, this is essential for the implementation of a first-stage working system. The current remodelling is optimised for jazz, but generally the aim is to develop a set of core rules that explain basic harmony structures which can be augmented with style specific rules. The grammar incorporates four levels: a phrase level, functional level, scale degree level and surface level. The phrase level divides a piece into phrases, the functional level specifies the functional role a certain scale degree has within a phrase. The scale degree captures the relation between the chord and the key and the surface level expresses the actual chord with all its possible additions, inversions, etc. An upper-case scale degree denotes a major and a lower-case scale degree denotes a minor chord.

Below we give an outline of the model by listing the main context-free rules. A piece always consists of one or more phrases (P). The phrase level of the grammar distinguishes two types of phrases: phrases which end on a perfect cadence (PCP) and phrases which end with a half-cadence (HCP). Perfect cadence phrases are distinguished by ending with a tonic function (t) upon which all subordinate harmonic elements are dependent, whereas half-cadence phrases force a phrase to end with a dominant function (d) which results in a tonicization of, or a perfect or imperfect cadence on the dominant. The +-sign denotes a possible repetition.

1. $Piece \rightarrow P_+$
2. $P \rightarrow PCP$

3. $P \rightarrow HCP$

At the functional level, the grammar encapsulates core relationships between the three main harmonic functions: tonic (t), dominant (d) and subdominant (s). A tonic (t) may be preceded by a subordinate dominant (d), and a dominant may be preceded by a subordinate subdominant (s).

4. $PCP \rightarrow d\ t_{+} \mid d\ d\ t_{+} \mid t\ d\ t$
5. $HCP \rightarrow t_{+}\ d$
6. $d \rightarrow s\ d$

Functional elements, i.e. tonics, dominants, or subdominants, can translate into multiple scale degrees, as several chords can fulfil the same harmonic functions. These functional rules may be applied recursively, but finally all rules will translate into scale degrees. Rule 10 and rule 11 enable the derivation of scale degrees that are parallels of the tonic (tpg).

7. $t \rightarrow$ I
8. $d \rightarrow$ V | vii^{0}
9. $s \rightarrow$ ii | IV
10. $t \rightarrow tpg$
11. $tpg \rightarrow$ vi | iii

The functional level also incorporates a number of additional prolongational rules that allow for the derivation of more distant harmonic structures such as the preparatory use of iii and tritone substitutions. Rule 12 incorporates a feature specifically designed for modelling the prototypical ii-V-I sequences in jazz harmony that are less prominent in other styles.

12. $x \rightarrow$ V(x) x | ii(x) V(x) x for any scale degree x
13. IV $\rightarrow$ iii IV
14. V(x) $\rightarrow$ bII(x) for any scale degree x

At the surface level scale degree nodes are translated into the actual surface chord label. These translation steps are straightforward when the key is known beforehand. For instance, a VI symbol in the key of C minor would translate into an A♭-chord. In addition, chord extensions are added at this level of description: a surface realisation of a VI chord may result in a $A\flat^6$ chord. Some of these surface extensions of chords are tied to their structural functions, e.g. the addition of the minor seventh in a D^7 chord label indicates a dominant function.

3.2.1 Implementation and parsing

There are some additional rules that have been implemented, but are not described here. Among these are rules for typical voice-leading and prolongational structures and some typical borrowings from the parallel key. Since we have not incorporated modulation yet, it is necessary to label these phenomena to be able to explain the remainder of the piece. Furthermore, there are rules that deal with typical well-known diminished chord transitions in various descending and ascending forms.

The grammar as specified above is not strictly a context-free grammar, because rule 12 and rule 14 use a variable binding. However, by expanding a rule for every scale degree x that it holds, a set of context-free rules can be created that yields the exact same outcome. Having a context-free grammar, a free Java implementation (Martin 2012) of an Earley Parser (Earley 1970) is used to parse the chord sequences in $O(n^3)$ time, where n is the number of chords.

Context-free grammars often create multiple parse trees. To select the optimal parse tree out of the set of parse trees, we provided the rules with weights (set by hand) and defined the total weight of a parse tree as the product of the weights of the rules used in its construction. Because of this, some rules have less chance to be used in the final derivation. This allows to select the best tree from the ambiguous parse trees by sorting the tree by their total weight. The complete grammar as well as the lead-sheets of the examples in Figure 3.1 are available online[13].

3.3 Common hierarchical structures

In this section we present six distance measures for the parse trees generated by the grammar discussed in the previous section. For the comparison of parse trees, we propose a solution based on the problem of tree inclusion, which is elaborately dealt

[13] `http://give-lab.cs.uu.nl/music/`

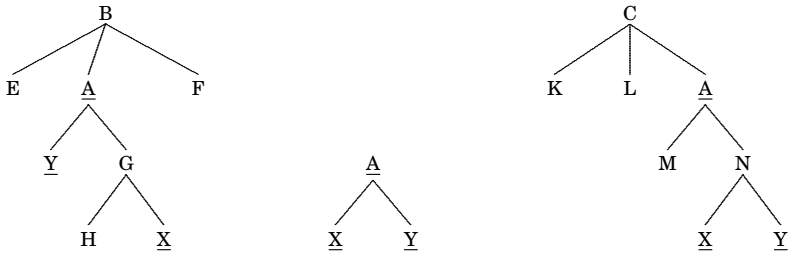


Figure 3.2: An example of a tree rooted by A that can be embedded into two larger trees rooted by B and C. The included nodes are underlined.

with in (Kilpeläinen 1992). Given the parse trees of two songs, we propose to search for the collection of labelled largest common embeddable subtrees (LCESs) for every combination of phrases. The LCES is the largest tree that is included in both parse trees. This means that there exists a one-to-one mapping from the nodes of the LCES to the nodes with the same label in both parse trees that preserves ancestor relations, but not necessarily parent relations. When processing harmony parse trees, this is a natural thing to do because typically a chord progression is augmented by adding a structure to the left branch and repeating the right branch, e.g. when a Dm is prepared by an A^7 chord. Hence, if both trees are similar, the LCES reflects the structure of the parse trees it is generated from, and if both trees are dissimilar, the resulting LCES will be much smaller and have a structure different from the structure of the trees it has been created from. In the next sections we explain the calculation of the LCES, and how we use it to define six distance measures.

3.3.1 Preliminaries

A rooted tree is a structure denoted with $P = (V, E, P_{root})$, where V is a finite set of nodes, E is the set of edges connecting nodes in V, and P_{root} is the root of the tree P. The nodes of the parse trees generated by the grammar of Section 3.2 are all labelled and the label of node v is denoted with $label(v)$. The subtree of P rooted at node v is denoted with P_v and $children(v)$ denotes the subset of V with nodes that have v as parent. Similarly, $desc(v)$ denotes the decedents of v, i.e. the subset of V that is contained in P_v and have v as ancestor. Furthermore, we use some additional functions. The function $po(v)$ denotes the post order number that is assigned to a node v in a postorder traversal. $depth(P)$ denotes the depth of a tree, i.e. the number of nodes in the longest path from leaf to the root. Finally, the $degree(P)$ is the degree of a tree, i.e. the maximum number of children.

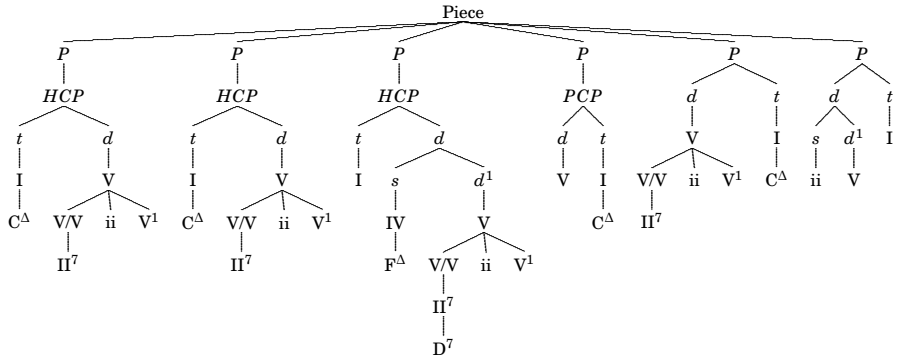


Figure 3.3: The Combined LCES for every combination of phrases of the parse trees of *Take the 'A' Train* as in Figure 3.1.

We say that a tree $P = (V, E, P_{root})$ is *included* in a tree $T = (W, F, T_{root})$ if there exists an embedding of P into T. An embedding is an injective function f, mapping each node in P to a node in T, that preserves labels and ancestorship (not necessary parentship). Figure 3.2 shows an example of an included tree. Note that a left-to-right ordering of the descendants is not required. All can be captured in the following definition:

Definition 3.3.1. The tree P is embedded a tree T if there exists a function $f : V(P) \rightarrow V(T)$, and for every node u in P, v in T the following holds:

1. $f(u) = f(v)$ if and only if $u = v$,
2. $label(u) = label(f(u))$,
3. u is an ancestor of v in P if and only if $f(u)$ is an ancestor of $f(v)$ in T.

3.3.2 Labelled largest common embeddable subtrees

We are not aware of an algorithm that calculates the largest common embeddable subtree for labelled trees. Gupta and Nishimura (1998) have developed a $O(n^{2.5} \log n)$ time algorithm for finding this tree for two unlabelled trees. The algorithm we present here calculates the largest common embeddable subtree for the labelled case and expands on the general tree matching ideas as described in (Kilpeläinen 1992, ch. 3).

Algorithm 3.1 calculates the LCES of two trees $P = (V, E, P_{root})$ and $T = (W, F, T_{root})$. To store the nodes of the subtrees of the LCES the algorithm uses a table M such that $M[po(w)]$ stores the subtrees that can be embedded into both P and T_w. The algorithm

Algorithm 3.1 Labelled largest common embeddable subtree

```
1: procedure LCES(P, T)
2:     M ← ∅
3:     for all w ∈ W in postorder do
4:         for all v ∈ V in postorder do
5:             if label(v) = label(w) then
6:                 x ← new node
7:                 label(x) ← label(v)
8:                 if children(w) = ∅ then
9:                     add x to M[po(w)]
10:                else
11:                    for all w′ ∈ children(w) do
12:                        for all x′ ∈ M[po(w′)] do
13:                            if x′ ∈ desc(v) then
14:                                add (x, x′) to M[po(w)]
15:                            else
16:                                add x′ to M[po(w)]
17:                            end if
18:                        end for
19:                    end for
20:                    add x to M[po(w)]
21:                end if
22:            end if
23:        end for
24:        if M[po(w)] = ∅ then
25:            for all w′ ∈ children(w) do
26:                add M[po(w′)] to M[po(w)]
27:            end for
28:        end if
29:    end for
30:    return M[po(T_root)]
31: end procedure
```

builds the LCES up from the leaves by traversing the nodes of T and P in postorder. When a node v with an identical label as w is encountered (line 5), the algorithm creates a new node x with the same label as v. In case w is a leaf, x is stored in M (lines 8–9). In case w is an internal node, we look up the subtrees in M that match the children of w. Because the tree is processed in postorder these nodes were previously stored in M and can be retrieved from $M[po(w')]$ for each child w'. If a previously stored subtree rooted by x' is a descendant of v, this subtree becomes a child of the new node x, by adding a new edge (x,x') to $M[po(w)]$ (lines 10–15). Otherwise, if x' is not a descendant of v, x' is stored in $M[po(w)]$ (line 16). After all, a common ancestor can show up in a next iteration. Finally, the new subtree x is stored in M as well (line 20). If the label of w does not match any of the labels of the nodes in P, the subtrees stored in M for all children w' of w are added to $M[po(w)]$ (lines 24–28). This process continues until all nodes of T have been matched against all nodes of P and finally $M[po(T_{root})]$, the LCES of P and T, is returned. A drawback of our algorithm is that it is incapable of dealing with duplicate labels. Therefore we number the duplicate labels that descent the same phrase.

The running time of the algorithm is dominated by the lines 3–23. For each of the $O(nm)$ combinations of w and v (lines 3 and 4) a constant time test is performed. Because the labels are unique, only $\min(n,m)$ times each of the $O(n)$ nodes in the subtrees that has been stored in $M[po(w)]$ so far (line 12), is checked against each of the $O(m)$ descendants of node v (line 13). This results in a time complexity for the whole algorithm of $O(\min(n,m)nm)$.

3.3.3 Distance measures

We base the distance measures on the LCES, but we do not calculate the LCES of two parse trees directly for two reasons. First, as we can see in Figure 3.1, there are quite some duplicate labels in the parse trees which our algorithm cannot handle. Second, if a parse tree of a song contains a repetition of a phrase that the matched song does not have, the repeated phrase cannot be matched. To solve these two problems we compare parse trees in a phrase-wise fashion. For every phrase in the target tree T we calculate the LCES for every phrase of the pattern parse tree P and pick the largest LCES to create a *Combined LCES* (see Figure 3.3). The duplicate labels are re-labelled per phrase too in preorder (see the subscripts in Figure 3.1 and 3.3). Because the number of duplicate labels per phrase is small, the labellings will be nearly identical if two trees have a similar harmonic structure. Note that if the two parse trees T and P have a different number of phrases, the structure of the Combined LCES will differ if P is used as a target tree, because for every phase in the T a LCES is constructed. This makes every Combined LCES based measure is asymmetrical.

We propose three distance measures for sequences of symbolic chord labels based on the Combined LCES:

1. *Relative Combined LCES Size Distance (Size).* By dividing the number of nodes in the target tree T by the number of nodes in the Combined LCES a distance measure between 1 and 0 is obtained that is normalised by the size of T.

2. *Grammar Violation Distance (Violation).* If two trees are not similar, the combined LCES will contain connections between nodes that cannot be explained by the grammar. By dividing the number of nodes in the target tree T—which are grammatical by definition—by the number of grammatical nodes in the Combined LCES we obtain a distance measure between 1 and 0 that is normalized by the size of T.

3. *Average Depth Distance (Depth).* If two chord progressions use the same chords, but have a very different harmony structure, this will be reflected in their harmony trees. If these harmony trees are matched, their Combined LCES will contain the matching nodes, but it will have a high degree and a limited depth. Hence, we also use the depth of the Combined LCES as a measure of similarity. By dividing the average leaf depth of T by the average leaf depth of the Combined LCES, we obtain a distance measure between 1 and 0, that is normalized by the size of T.

One can observe in Figure 3.1 that, having the actual parse tree structure, the actual chord labels are not of much importance any more. Given two similar sequences, it is rather arbitrary whether the chords labels match or not: the structure of the harmony determines the similarity. Therefore we can remove each leaf node describing a surface chord from the Combined LCES and target trees. The structure of the phrase, functional and scale degree level remains unchanged. As a consequence, this yields three additional harmonic distance measures that are concerned with the structure of the harmony only. Other Combined LCES distance measures can be thought of.

3.4 Experiment

We have evaluated the six LCES based distance measures described in the previous section in an experiment. We assembled a dataset of 72 symbolic chord label sequences extracted from user-generated Band-in-a-Box files that were collected on the Internet. Band-in-a-Box is a software package that generates accompaniment given a certain chord sequence provided by the user. This dataset consists of 51 different pieces of

	With Chord Labels			*Without Chord Labels*			
Distance:	Size	Violation	Depth	Size	Violation	Depth	Edit Distance
MAP:	0.79	0.81	0.72	0.81	0.86	0.73	0.67

Table 3.1: The MAP of the six Combined LCES based similarity measures and a baseline edit distance.

which 16 pieces contain two or three versions, forming 16 song classes. These pieces are all jazz standards from before the 1970's and can all be found in the Real Book (2005) or similar sources. All parse trees of these pieces are available online[14]. The task is to retrieve the other versions of a song class, given a certain query song from that class. All songs containing more than one version are used as a query and the rankings are analysed by calculating the mean average precision (MAP). To place the results in perspective, we calculate the edit distance (Levenshtein 1966) between all chord sequences, represented as a string of chord labels, as a baseline measure.

The results are presented in Table 3.1 and indicate that all Combined LCES based methods perform better than the baseline edit distance. The results can be considered good, and show that especially the number of grammatical connections in the Combined LCES is a promising indicator for harmonic similarity. However, the dataset used is too small to draw far-reaching conclusions. Nevertheless, the experiment does show that a using hierarchical musical knowledge about the harmony structure is a promising direction of research.

3.5 Discussion

In this chapter we introduced a new approach to harmonic similarity. We showed that a grammar of tonal harmony can be adapted in such a way that it is usable for matching harmony sequences. However, there are some open issues that need to be addressed. The most hampering problem is that we cannot calculate harmonic analysis based distance measures if one of the pieces does not parse and for every context-free grammar there are always pieces imaginable that do not parse, especially when an automatic chord transcription front-end is used and a margin of error should be accounted for. After all, noise or other obscurations in the audio signal might frustrate the successful extraction of the right chord label. However, often this is a local problem and only one or two chords cannot be explained by the grammar. In Chapter 4 we solve this by using an error-correcting parser.

[14] `http://give-lab.cs.uu.nl/music/`

Another property of context-free grammars is that sequences can have multiple ambiguous parse trees. Although chord sequences can be ambiguous by nature and a grammar of tonal harmony should reflect some ambiguity, it may cause the number of solutions to explode in certain cases, especially if one considers the repetitive character that music can sometimes exhibit. Hence, we should try to constrain the derivation of ambiguous solutions as much as possible. In the next chapter we will extensively elaborate on how this can be achieved in an elegant manner. Also, the grammar as presented here features several problems with respect to the parsing of phrase boundaries. It is often unclear at the phrase level when a dominant marks the end of a halve cadence phrase and when it is part of a perfect cadence. Hence, the phrasing rules (4 and 5) constitute a main source of ambiguities. It seems that the harmony information alone does not contain enough information to correctly identify the phrase boundaries at all times. A set of additional preference rules can be designed for future versions of the model to rule out unlikely phrase-boundaries. These should be based on metrical information which is not yet incorporated in the present model.

What made the development of the grammar presented here less convenient was the fact that all rules needed to be written by hand. The XML specification describing the grammar as used in this chapter contains 1819 lines to describe 438 context-free rules. Especially the manual expansion of the secondary dominant rules (e.g. Rule 12) is a tedious and error-prone enterprise. If the aim is to have a flexible and maintainable model, some sort of meta-grammar is needed that will automatically expands rules like Rule 12. The HARMTRACE model presented in the next chapter will tackle this issue as well.

Also the matching of parse trees should be further investigated and here some issues remain open as well. First of all, to get a better impression of the retrieval performance of the similarity measures described in this chapter a larger scale comparison should be set up. Second, in the current matching approach the problem of duplicate labels was solved by relabelling the duplicate labels. This is clearly sub-optimal because it forces a first occurrence of a node to always match against the first occurrence of a node in the matched parse tree. Finally, it is unclear how the phrase-wise comparison affects the results of the matching. All will be addressed in Chapter 5.

The research presented in this chapter demonstrates how a prototypical grammar of harmony may characterise harmonic similarity in a musical way. This research provides some first preliminary evidence that demonstrates the importance of cognitive and theoretic models of music in the design of appropriate methods for MIR tasks. In the following chapters we will build on the ideas presented this chapter and propose solutions for the issues raised. In Chapter 4 we take harmony analysis to the next level and solve the parsing problems that we exposed in this chapter. Next, in Chapter 5,

we demonstrate how such a model of tonal harmony can have a large impact on the quality of the representation, analysis and retrieval of tonal music.

Chapter 4

Automatic functional harmonic analysis

FOR ages, musicians, composers, and musicologists have been theorising the structure of music to better understand how music is perceived, performed, and appreciated. In particular, tonal harmony exhibits a considerable amount of structure and regularity, and the first theories describing tonal harmony date back at least to the 18th century (Rameau 1971). Since then, a rich body of literature has emerged that aims at explaining the harmonic regularities in both informal and formal models (e.g., Lerdahl and Jackendoff 1996). Such models have attracted numerous computer music researchers to automate the analysis and generation of harmony. However, most of these theories have proven to be very hard to implement (e.g., Clarke 1986). Not in the least place, the prototypical implementation presented in Chapter 3 showed that implementing a model of tonal harmony as a context-free grammar, albeit very useful and promising, is far from trivial. Currently, we are not aware of a model that has a working implementation that effectively analyses tonal harmony and deals robustly with noisy data, while remaining simple and easy to maintain, and scaling well to handle musical corpora of considerable size. In this chapter we present HARMTRACE,[15] a system that meets these requirements using state-of-the-art functional programming techniques. HARMTRACE allows to easily adapt the harmonic specifications, empirically evaluate the harmonic analyses, and use these analyses for tasks such as similarity estimation and automatic annotation of large corpora.

The HARMTRACE harmony model draws on the ideas of Rohrmeier (2007, 2011). Rohrmeier modelled the core rules of Western tonal harmony as a (large) context-free grammar (CFG, Chomsky 1957). Later, De Haas et al. (2009) implemented this grammar and specifically tuned it for jazz harmony, with the aim of modelling harmonic similarity (see also Chapter 3). The HARMTRACE system transfers these ideas to a functional programming setting, solving typical problems that occur in context-free parsing, e.g. the rejection of non-parsing pieces, and controlling the number of am-

[15] Harmony Analysis and Retrieval of Music with Type-level Representations of Abstract Chord Entities

biguous solutions. Since it relies on advanced functional programming techniques not readily available in most programming languages, HARMTRACE is inextricably bound to Haskell (Peyton Jones 2003). Haskell is a purely functional programming language with strong static typing. It is purely functional because its functions, like regular mathematical functions, guarantee producing the same output when given the same input. It is strongly typed because it enforces restrictions on the arguments to functions, and it does so statically, i.e. at compilation time. Through its main implementation, the Glasgow Haskell Compiler,[16] Haskell offers state-of-the-art functional programming techniques, like error-correcting combinator parsers, type-level computations, and *datatype-genericity* (polytypic functions) that are not available in any other mainstream language.[17] These features proved to be essential to HARMTRACE, as we will show.

Following Rohrmeier, a core assumption that underlies our harmony model is that Western tonal harmony is organised hierarchically and transcends Piston's table of usual root progressions (Piston and DeVoto 1991, ch. 3, p. 21). As a consequence, within a sequence of chords some chords can be removed because of their subordinate role, leaving the global harmony structure intact, while removing other chords can significantly change how the chord sequence is perceived. This is illustrated in the sequence displayed in Figure 4.1: the D^7 chord in this sequence can be removed without changing the general structure of the harmony, while removing the G^7 or the C at the end would cause the sequence to be perceived very differently. This implies that within a sequence not all chords are equally important, and must be organised hierarchically. This hierarchical organisation is reflected in the tree in Figure 4.1. The subdominant F has a subordinate role to the dominant G^7, which is locally prepared by a secondary dominant D^7. The tonic C releases the harmonic tension built up by the F, D^7, and G^7.

As in the Chapter 3, the development of HARMTRACE has been driven by its application in content-based Music Information Retrieval (MIR, Downie 2003) research. Within MIR the notion of *musical similarity* plays a prominent role because it allows ordering musical pieces in a corpus. Using such an ordering, one could imagine retrieving harmonically related pieces of music, like cover-songs, classical variations, or all blues pieces in a corpus. For performing such searches, a measure of harmonic similarity is essential. De Haas et al. (2009, 2011b) and Magalhães and De Haas (2011) show

[16] `http://www.haskell.org/ghc/`

[17] Of course, any Turing-complete language can be used to implement a particular program; this also holds for HARMTRACE. However, most programming languages have a much simpler type system, and cannot be used to do type-indexed computations. Other non-mainstream programming languages that do feature the same expressiveness, as explicitly used in the HARMTRACE's Haskell implementation, are Coq and Agda. We prefer these languages over, for instance C, Java, or LISP, because their strong type system gives us good guarantees of correctness, make the development process clearer and easier, and allow for compiler type-directed optimisations (e.g. fusion laws).

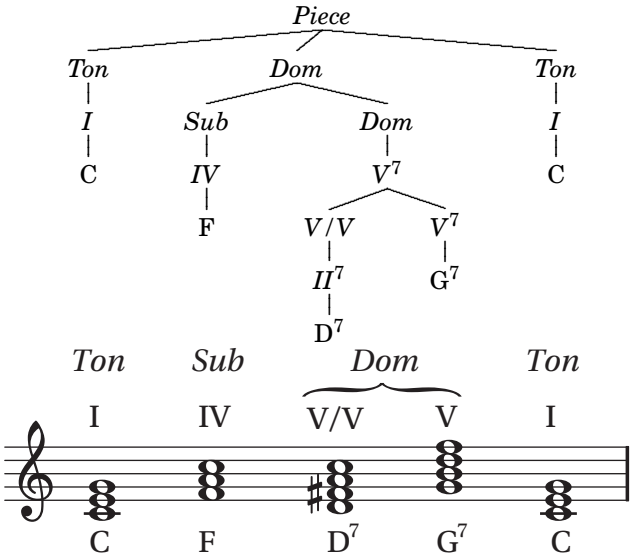


Figure 4.1: A typical chord sequence and its harmonic analysis (as generated by HARMTRACE) The chord labels are printed below the score, and the scale degrees and functional analysis above the score. We ignored voice-leading for simplicity.

that analysing the hierarchical relations between chords in a sequence significantly improves the quality of a harmonic similarity measure in a retrieval task (see also Chapter 5). The application to MIR explains some of the choices made in the development of HARMTRACE. In particular, because a large corpus of mainly jazz chord sequences is available for retrieval tasks, the harmony model exhibits a bias towards jazz harmony.

A fully-functional model of tonal harmony that can quickly analyse chord sequences offers several benefits. First, musicologists study the harmonic structure of pieces and annotate them by hand. This is a time-consuming enterprise, especially when large corpora are involved. With the automatic annotation techniques that we present in this chapter, this can be done quickly, even for large corpora possibly containing errors. Second, because our system captures the global and local relations between chords, it can be used to improve systems designed for other tasks that can benefit from this contextual information, such as chord labelling systems. These systems traditionally determine the chord root and type based on the musical information (audio or notation) from within a limited time frame, without incorporating knowledge about the surrounding chord sequences. In Chapter 6 we elaborate on this in detail. Last, HARMTRACE could aid in (automatic) composition by generating sequences of chords, or

by generating harmonically realistic continuations given a sequence of chords.

Contribution. We present a functional model of Western tonal harmony. In contrast to other formal approaches to tonal harmony, we present an implemented system that, given a sequence of chord labels, automatically derives the functional dependency relations between these chords. We elaborate on how we can tackle the problems raised in Chapter 3 by exploiting some advanced functional programming techniques available in the functional programming language Haskell. Our system is fast, easy to modify and maintain, and robust against noisy data.

This chapter is organised as follows. We start by discussing a relevant selection of the large body of existing literature on harmony theory and modelling in the next section. In Section 4.2 we explain our harmony model and in Section 4.3 we elaborate on some of the implementation details. Subsequently, we evaluate some detailed example analyses created by our model in Section 4.4. Next, in Section 4.5 we show that HARMTRACE can deal with large amounts of noisy data. Finally, we conclude this chapter by discussing the limitations of our system, and pointing out the future directions of our research.

4.1 Related work

The nineteenth and twentieth century have yielded a wealth of theoretical models of Western tonal music; in particular, tonal harmony has been prominently researched. Most theories that describe the relationships between sequential chords capture notions of order and regularity; some combinations of chords sound natural while others sound awkward (e.g., Rameau 1971). These observations led music theorists to develop ways to analyse the function of a chord in its tonal context (e.g., Riemann 1893). Unfortunately, the majority of these theories are formulated rather informally and lack descriptions with mathematical precision. In this section we give a condensed overview of the theories that played an important role in the formation of the harmony model we present in this paper.

Seminal has been the *Generative Theory of Tonal Music* (GTTM, Lerdahl and Jackendoff 1996) that further formalised the ideas of Schenker (1935). Lerdahl and Jackendoff structured Western tonal compositions by defining recursive hierarchical dependency relationships between musical elements using well-formedness and constraint-based preference rules. The GTTM framework distinguishes four kinds of hierarchical structure: meter, grouping, time-span reduction, and prolongational reduction. Although GTTM can be considered one of the greatest contributions to music theory and music cognition of the last decades, implementing the theory is difficult be-

cause the often vague and ambiguous preference rules lead to a wide range of possible analyses (Clarke 1986; Temperley 2001, ch. 1; Hamanaka et al. 2006).

The recursive formalisation proposed by Lerdahl and Jackendoff suggests a strong connection between language and music. Also, many other authors have argued that tonal harmony should be organised in hierarchical way similar to language, leading to numerous linguistically-inspired models since the 1960's (Roads 1979). One of the pioneers to propose a grammatical approach to harmony was Winograd (1968). More recently, Steedman (1984, 1996) modelled the typical twelve-bar blues progression with a categorial grammar; Chemillier (2004) elaborates on these ideas by transferring them to a CFG. Similarly, Pachet (1999) proposes a set of rewrite rules for jazz harmony comparable to Steedman's grammar. Pachet furthermore shows that these rules can be learned from chord sequence data in an automated fashion. Additionally, quasi-grammatical systems for Schenkerian analysis have been proposed recently (Marsden 2010). Furthermore, Choi (2011) developed a system for analysing the harmony of jazz chord sequences; this system identifies common harmonic phenomena, like secondary dominants and tritone substitutions, and labels the chords involved accordingly. The relation between music and language is not merely a theoretical one; a growing body of evidence, also from neuropsychology and neuroimaging, suggests that music and language are more closely related than was previously believed (Patel 2003; Koelsch 2011). However, an in depth overview of the animated debate on the relation between music and language is beyond the scope of this thesis.

The generative formalism proposed by Rohrmeier (2007, 2011), which the HARMTRACE model greatly draws on, expands these earlier approaches in a number of ways. Rohrmeier gives an encompassing account of how tonal harmony relationships can be modelled using a generative CFG with variable binding. It aims to model form, phrasing, theoretical harmonic function (Riemann 1893), scale degree prolongation (Schenker 1935; Lerdahl and Jackendoff 1996), and modulation. Rohrmeier's grammar differs from earlier grammatical formalisms in various ways. Steedman's approach (Steedman 1984, 1996) merely concerns blues progressions. It features seven context-sensitive rules (with variations), but it omits a number of theoretically important features to support broader domains. Rohrmeier's formalism also differs from GTTM: GTTM aims at describing the core principles of tonal cognition, and harmony is covered mainly as a prolongational phenomenon, while Rohrmeier's formalism describes the structure of tonal harmony from an elaborate music-theoretical perspective with concrete context free rules. Rohrmeier acknowledges that a full account of tonal harmony would require a large number of varying style-specific rules, and his formalism aims to capture only the core rules of Western tonal harmony.

In Chapter 3 we performed a first attempt to implement the ideas of Rohrmeier. Al-

though the results were promising, the used context-free parsing techniques hampered both theoretical as well as practical improvements. First, a sequence of chords that does not match the context-free specification precisely is rejected and no information is given to the user. For example, appending one awkward chord to an otherwise grammatically correct sequence of chords forces the parser to reject the complete sequence, not returning any partial information about what it has parsed. Second, musical harmony is ambiguous and chords can have multiple meanings depending on the tonal context in which they occur. This is reflected in all grammatical models discussed above. A major drawback of context-free grammars is that they are very limited in ways of controlling the ambiguity of the specification. It is possible to use rule-weightings and to set low weights to rules that explain rare phenomena. This allows for ordering the ambiguous solutions by the total relevance of the used rules. However, this does not overcome the fact that, for some pieces, the number of parse trees grows exponentially, given the number of input chords. Last, writing context-free rules by hand is a tedious and error-prone enterprise, especially since the grammatical models can become rather large. For instance, a rule generalising over an arbitrary scale degree has to be expanded for each scale degree, I, II, III, etc. Hence, some form of high-level grammar generation system is needed to allow for generalising over scale degree and chord type, and to control conditional rule execution.

Another important model that has influenced the development of HARMTRACE is that of Temperley (2001) and Temperley and Sleator (1999). They give an elaborate formal account of Western tonal music, and also provide an efficient implementation. This rule-based system, which is partly inspired by GTTM, can perform an analysis of the chord roots and the key given a symbolic score, but does not formalise the hierarchical relations between chords. Our system continues where Temperley's left off: we assume we have the chord and key information of the piece, and model the global and local dependency relations between these chords. Hence, the input to our model consists of plain-text key and chord label information. In Chapter 6 we demonstrate a chord transcription prototype, which can be used directly as front-end for the methods presented in the other chapters of this thesis.

4.2 The HarmTrace system

In this section we explain how we model the regularities and hierarchical dependencies of tonal harmony. HARMTRACE transfers the ideas of the previous chapter to a functional setting. While the contributions of the majority of models we discussed in the previous section are purely theoretical, we present a system that can be evaluated empirically and is usable in practice. However, this comes at a price: our present

model does not support full modulation, and can only distinguish between parallel keys—going from major to minor or vice versa without changing the root of the key. As a consequence, this requires the model to have information about the key of the piece. Also, since we mainly use jazz-oriented input data in this article, we also include some specific jazz harmony specifications. Figure 4.2 shows an example analysis as produced by HARMTRACE. The chords that were used as input are the leaves of the tree, and the internal nodes represent the harmonic relations between the chords.

Music, and harmony in particular, is intrinsically ambiguous; certain chords can have multiple meanings within a tonal context. Although a model of tonal harmony should reflect some ambiguity, defining many ambiguous specifications can make the number of possible analyses grow exponentially for certain chord progressions. However, in most of the ambiguous cases it is clear from the context which of the possible solutions is the preferred one. Hence, we can select the favoured analysis by constraining the application of the specification leading to the undesired analysis. In cases where it is less clear from the context which solution is preferred, we accept a small number of ambiguous analyses.

The HARMTRACE system explores the relations between (generalised) algebraic data types and context-free production rules. A CFG defines a language: given a set of production rules and a set of words or *tokens*, it accepts only combinations of tokens that are valid sequences of the language. The notion of an algebraic datatype is central in Haskell. Similar to a CFG, a datatype defines the structure of values that are accepted. Hence, a collection of datatypes can be viewed as a very powerful CFG: the type-checker accepts a combination of values if their structure matches the structure prescribed by the datatype, and rejects this combination if it does not. Within HARMTRACE, the datatypes represent the relations between the structural elements in tonal harmony, and the chords are the values. However, an important difference between a CFG and a Haskell datatype is that datatypes provide more much modelling freedom and control, especially the *generalised* algebraic datatypes (Schrijvers et al. 2009) that we use. They allow constraining the number of applications of a specification, constraining the conditions for application, and ordering specifications by their importance. This allows defining mode and key-specific specifications, excluding scale degree-specific applications (e.g. Spec. 18) of transposition functions, and preferring certain specifications over others. For technical details, we refer to Magalhães and De Haas (2011) and the code online [18]

[18] HARMTRACE version 0.6 can be downloaded from `http://hackage.haskell.org/package/HarmTrace-0.6`

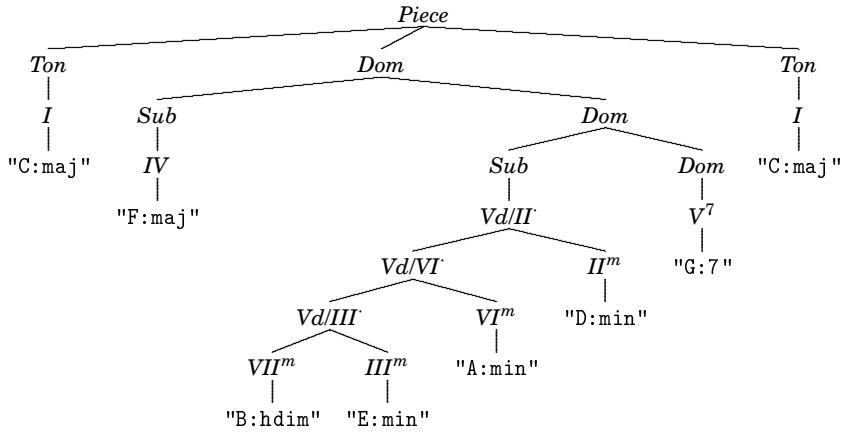


Figure 4.2: An example of a diatonic cycle of fifths progression in C major. The leaves represent the input chords and the internal nodes denote the harmony structure. *Ton*, *Dom*, and *Sub* denote tonic, dominant, and subdominant. The *Vd/X* nodes represent diatonic fifth successions.

4.2.1 A model of tonal harmony

We now elaborate on how our harmony datatypes are organised. Haskell knowledge is not required to understand our model: we use a simplified syntax to describe the datatype specifications that is inspired by the syntax used to describe CFGs. We start by introducing a variable (denoted in upper case with bold font) **m** for the mode of the key of the piece, which can be major or minor. The mode variable is used to parametrise all the specifications of our harmonic datatype specification; some specifications hold for both modes (**m**), while other specifications hold only for the major (Maj subscript) or minor mode (Min). Similar to a CFG, we use a | to denote alternatives, and a + to represent optional repetitions of a datatype.

1 $Piece_{\mathbf{m}} \rightarrow Func^{+}_{\mathbf{m}}$
2 $Func_{\mathbf{m}} \rightarrow Ton_{\mathbf{m}} \mid Dom_{\mathbf{m}}$ $\quad \mathbf{m} \in \{\text{Maj}, \text{Min}\}$
3 $Dom_{\mathbf{m}} \rightarrow Sub_{\mathbf{m}}\, Dom_{\mathbf{m}}$

Spec. 1–3 define that a valid chord sequence, $Piece_{\mathbf{m}}$, consists of at least one and possibly more functional categories. A functional category classifies chords as being part of a tonic ($Ton_{\mathbf{m}}$), dominant ($Dom_{\mathbf{m}}$), or subdominant ($Sub_{\mathbf{m}}$) structure, where a subdominant must always precede a dominant. These functions constitute the top-level categories of the harmonic analysis and model the global development of tonal tension:

a subdominant builds up tonal tension, the dominant exhibits maximum tension, and the tonic releases tension. The order of the dominants and tonics is not constrained by the model, and they are not grouped into larger phrases.

$$
\begin{array}{lll}
4 & Ton_{\text{Maj}} & \rightarrow I_{\text{Maj}} \quad | I_{\text{Maj}}\, IV_{\text{Maj}}\, I_{\text{Maj}} \,|\, III^{m}_{\text{Maj}} \\
5 & Ton_{\text{Min}} & \rightarrow I^{m}_{\text{Min}} \quad | I^{m}_{\text{Min}}\, IV^{m}_{\text{Min}}\, I^{m}_{\text{Min}} \,|\, III\flat_{\text{Maj}} \\
6 & Dom_{\mathbf{m}} & \rightarrow V^{7}_{\mathbf{m}} \quad | V_{\mathbf{m}} \\
7 & Dom_{\text{Maj}} & \rightarrow VII^{m}_{\text{Maj}} \quad | VII^{0}_{\text{Maj}} \qquad \mathbf{c} \in \{\varnothing, m, 7, 0\} \\
8 & Dom_{\text{Min}} & \rightarrow VII\flat_{\text{Min}} \\
9 & Sub_{\mathbf{m}} & \rightarrow II^{m}_{\mathbf{m}} \\
10 & Sub_{\text{Maj}} & \rightarrow IV_{\text{Maj}} \quad | III^{m}_{\text{Maj}}\ \ IV_{\text{Maj}} \\
11 & Sub_{\text{Min}} & \rightarrow IV^{m}_{\text{Min}} \quad | III\flat_{\text{Min}}\, IV_{\text{Min}}
\end{array}
$$

Spec. 4–11 translate the tonic, dominant, and subdominant datatypes into scale degree datatypes. A tonic translates to a first degree, a dominant to a fifth degree, and the subdominant to a fourth degree in both major and minor keys. We denote scale degree datatypes with Roman numerals, but because our model jointly specifies datatypes for major as well as minor mode, we deviate from notation that is commonly used in classical harmony and represent scale degrees as intervals relative to the diatonic major scale. For example, III^{m} unequivocally denotes the minor chord built on the note a major third interval above the key's root and does not depend on the mode of the key. Similarly, a $III\flat$ denotes a major chord built on the root a minor third interval above the key's root.

A scale degree datatype is parametrised by a mode, a chord class, and the scale degree, i.e. the interval between the chord root and the key root. The chord class categorises scale degrees as one of four types of chords (denoted in lower case with superscripts) and is used to constrain the application of certain specifications, e.g. Spec. 16–17. The four classes are major (no superscript), minor (m), dominant seventh (7), and diminished (0). Chords in the minor class contain a minor or diminished triad and can have possible altered or non-altered additions, except for the diminished seventh. Chords categorised as major contain a major triad and can be extended by with non-altered additions, with exception of the dominant seventh chord (with additions). Chords of the dominant class have a major or augmented triad and a minor seventh and can be extended with altered or non-altered notes. Finally, the diminished class contains only the diminished seventh chord. In case a specification holds for all chord classes, the chord class variable $\mathbf{c}$ is used. This allows us to define certain specifications that only

hold for dominant chords, while other specifications might hold only for minor chords, etc.

Tonics can furthermore initiate a plagal cadence (Spec. 4–5). We deliberately chose not to model the plagal cadence with scale degrees and not with *Sub* and *Ton* because this keeps the number of possible analyses smaller. Also a $Sub_{\mathbf{m}}$ can translate into the a $II^{m}_{\mathbf{m}}$, because of its preparatory role, and the dominant translates into the seventh scale degree, VII^{7}_{Maj} (and $VII\flat^{7}_{\text{Min}}$ in minor). Similarly, we could have chosen to model the $Sub_{\mathbf{m}}$ to translate also to the VI^{m}_{Maj} ($VI\flat_{\text{Maj}}$ in minor). However, we chose to solve this by creating specifications for chains of diatonic fifths instead (Spec. 18–19, see for instance Figure 4.2).

The $Ton_{\mathbf{m}}$ resolving into III^{m}_{Maj} (and $III\flat_{\text{Min}}$ in minor), is perhaps the most unexpected transformation. Often the third degree can be explained as either a preparation of a secondary dominant (Spec. 17), as being part of diatonic chain of fifths (Spec. 19), or as supporting the subdominant (Spec. 10–11). However, in certain progressions it cannot be explained by any of these specifications, and is best assigned a tonic function since it has two notes in common with the tonic and does not really create any harmonic tension.

12 $I_{\text{Maj}} \rightarrow$ `"C:maj"|"C:maj6" |"C:maj7" |"C:maj9" |...`

13 $I^{m}_{\text{Min}} \rightarrow$ `"C:min"|"C:min7" |"C:min9" |"C:min(13)"|...`

14 $V^{7}_{\text{Maj}} \rightarrow$ `"G:7" |"G:7(b9,13)"|"G:(#11)"|"G:7(#9)" |...`

15 $VII^{0}_{\mathbf{m}} \rightarrow$ `"B:dim(bb7)"`

Finally, scale degrees are translated into the actual surface chords, which are used as input for the model. The chord notation used is that of Harte et al. (2005). The conversions are trivial and illustrated by a small number of specifications above, but the model accepts all chords in Harte et al.'s syntax. The model uses a key-relative representation; in Spec. 12–15 we used chords in the key of C. Hence, a I_{Maj} translates to the set of C chords with a major triad, optionally augmented with additional chord notes that do not make the chord minor or dominant. Similarly, V^{7}_{Maj} translates to all G chords with a major triad and a minor seventh, etc. To treat repeating chords in a natural way, we cluster chords with the same class and same scale degree, e.g. `"C:min7" "C:min9"`, in one datatype.

16 $X^{\mathbf{c}}_{\mathbf{m}} \rightarrow V/X^{7}_{\mathbf{m}}\, X^{\mathbf{c}}_{\mathbf{m}}$ $\qquad$ $\mathbf{c} \in \{\varnothing, m, 7, 0\}$

17 $X^{7}_{\mathbf{m}} \rightarrow V/X^{m}_{\mathbf{m}}\, X^{7}_{\mathbf{m}}$ $\qquad$ $X \in \{I, II\flat, II, \ldots, VII\}$

Besides these basic elements of tonal harmony, we distinguish various scale degree substitutions and transformations. For this we introduce the function V/X which transposes an arbitrary scale degree X a fifth up. Herewith, Spec. 16 accounts for the classical preparation of a scale degree by its secondary dominant, stating that every scale degree, independently of its mode, chord class, and root interval, can be preceded by a chord of the dominant class, one fifth up. Similarly, every dominant scale degree can be prepared with the minor chord one fifth above (Spec. 17). These two specifications together allow for the derivation of the typical and prominently present ii-V-I progressions[19] in jazz harmony. However, these specifications interfere with Spec. 4–11, causing multiple analyses. Because we prefer e.g. a II^m, V^7, and I to be explained as *Sub*, *Dom*, and *Ton*, we constrain the application of Spec. 16 and 17 to the cases were Spec. 4–11 do not apply.

18 $X^m_{\text{Maj}} \rightarrow V/X^m_{\text{Maj}}\, X^m_{\text{Maj}}$

19 $X_{\text{Min}} \rightarrow V/X_{\text{Min}}\, X_{\text{Min}}$

The model also accounts for the diatonic chains of fifths in major (Spec. 18) and minor (Spec. 19) keys. [20] These diatonic chain specifications are necessary to explain the typical *cycle of fifths* progressions: $I\ IV\ VII^m\ III^m\ VI^m\ II^m\ V^7\ I$ (see Figure 4.2 for the major case, and the *Autumn leaves* example of the next section in Figure 4.3). We constrain the major key specification to only apply to the minor chords because I, IV, and V^7 translate directly to *Ton*, *Sub*, and *Dom*, respectively. Similarly, Spec. 19 captures the same phenomenon in a minor key. Here, we restrict the application of the specification only to the major chords because, again, I^m, IV^m, and V^7 translate directly to *Ton*, *Sub*, and *Dom* in minor. Without these restrictions the parser would generate multiple solutions.

20 $X^7_{\mathbf{m}} \rightarrow V\flat/X^7_{\mathbf{m}}$

The harmony model in HARMTRACE allows for various scale degree transformations. Every chord of the dominant class can be transformed into its tritone substitution with Spec. 20. This specification uses another transposition function $V\flat/X$ which transposes a scale degree X a diminished fifth—a tritone—up (or down). This tritone substitution rule allows for the derivation of progressions with a chromatic baseline, e.g. Am $G\sharp^{7}$ G.

[19] the ii-V-I progression is a very common cadential chord progression which occurs frequently in jazz harmony, but also in other musical genres. The I-chord can practically be an arbitrary chord in a sequence which is preceded by its relative secondary dominant a fifth interval up, the V, and its relative sub dominant another fifth interval up, the ii.

[20] We implemented one exception to the diatonic minor specification that allows the $VI\flat$ to precede the II^m in minor. Here, the major chord precedes a minor chord. See $E\flat^{\Delta}$ $Am^{7\flat5}$ in Figure 4.3.

Because we want the application of the Spec. 16–20 to terminate, we limit the number of possible recursive applications of these rules (see parsing section below).

21 $X^0_{\mathbf{m}} \rightarrow III\flat/X^0_{\mathbf{m}}$
22 $X^7_{\mathbf{m}} \rightarrow II\flat/X^0_{\mathbf{m}}$

Diminished seventh chords can have various roles in tonal harmony. An exceptional characteristic of these chords—consisting only of notes separated by minor third intervals—is that they are completely symmetrical. Hence, a diminished seventh chord has four enharmonic equivalent chords that can be reached by transposing the chord by a minor third with the transposition function $III\flat/X$ (Spec. 21). In general, we treat diminished chords as dominant-seventh chords with a ♭9 and no root note. For instance, in a progression Am^7 $\mathrm{A}\flat^0$ G^7, the $\mathrm{A}\flat^0$ closely resembles the $\mathrm{G}^{7\flat9}$, because a $\mathrm{G}^{7\flat9}$ chord consists of G, B, D, F, and an $\mathrm{A}\flat^0$ chord consists of A♭, B, D, and F. This similarity is captured in Spec. 22, where $II\flat/X$ transposes a scale degree one semitone up. Similarly, by combining secondary dominants (Spec. 16) with Spec. 22, e.g. F $\mathrm{E}\flat^0$ ($\approx \mathrm{D}^{7\flat9}$) G, and an application of Spec. 21, e.g. F $\mathrm{F}\sharp^0$ ($\approx \mathrm{E}\flat^0$) G, we can account for most of the ascending diminished chord progressions. Within harmony theory this phenomenon is usually labelled as VII/X, where $\mathrm{F}\sharp^0$ would be the VII/X (in C major) in the previous example. However, since our model can explain it without a specific VII/X specification, there is no need to add one.

23 $Func_{\mathrm{Maj}} \rightarrow Func_{\mathrm{Min}}$
24 $Func_{\mathrm{Min}} \rightarrow Func_{\mathrm{Maj}}$
25 $Sub_{\mathrm{Min}} \rightarrow II\flat_{\mathrm{Min}}$

We support borrowings from the parallel key by changing the mode but not the root of the key in the $Func_{\mathbf{m}}$ datatype (Spec. 23 and 24). Although the parallel keys are often considered rather distantly related (there is a difference of three accidentals in the key signature), borrowing from minor in major occur frequently in jazz harmony. These specifications account, for instance, for the picardy third—ending a minor piece on a major tonic. The actual implementation of Spec. 23 and 24 differs marginally to overcome endless recurring transitions between major and minor. Finally, the Neapolitan chord $II\flat_{\mathrm{Min}}$ is categorised as being part of a Sub_{Min} structure (Spec. 25), which is also reachable in a major key through Spec. 23. Although it might be considered an independent musical event, it often has a subdominant function.

The datatype specification that we have presented in this section match the Haskell code closely. Nevertheless, to maintain clarity, some minor implementation details

were omitted; these can be found in the real datatype specifications of the model, i.e. the Haskell code as found online[21]

4.2.2 Parsing

Having a formal specification as a datatype, the next step is to define a parser to transform textual chord labels into values of our datatype. Writing a parser that parses labels into our datatype would normally mean writing tedious code that closely resembles the datatype specification. However, in Haskell we can use *datatype-generic programming*[22] techniques (Jeuring et al. 2009) to avoid writing most of the repetitive portions of the code. Moreover, we derive not only the parser automatically, but also a pretty-printer for displaying the harmony analysis in tree form, and functions for comparing these analyses. This makes the development and fine-tuning of the model much easier, as only the datatypes have to be changed, and the code adapts itself automatically. The technical details of the implementation our model and the used generic programming techniques are covered in the next section (see also: Magalhães and De Haas 2011).

Because music is an ever changing, culturally dependent, and extremely diverse art form, we cannot hope to model all valid harmonic relations in our datatype. Furthermore, songs may contain mistakes or mistyped chords, perhaps feature extraction noise, or malformed data of dubious harmonic validity. In HARMTRACE we use a parsing library featuring error-correction: chords that do not fit the structure are automatically deleted or preceded by inserted chords, according to the datatype structure (Swierstra 2009). The error-correction process uses heuristics to find a reasonable parse tree in a reasonable amount of time. For most songs, parsing proceeds with none or very few corrections. Songs with a very high error ratio denote multiple modulations, bad input, or a wrong key assignment. Note that depending on the severity of "unexpectedness" of a chord there might be multiple error-corrections necessary to create a valid analysis, e.g. one deletion and two insertions.

In our model, one particular parameter has a large influence on the parsing and error-correction process. This parameter controls the number of recursive applications of the specifications for secondary dominant and the like (Spec. 16–20). It must be set carefully, because setting it to a very low value will lead to bad analyses, while setting it to a high value will make the internally generated model very large, resulting in increased error-correction times and often sub-optimal error-correction solutions. For the examples and results in this paper we have used values between 5 and 7.

[21] `http://hackage.haskell.org/package/HarmTrace-0.6`

[22] Not to be confused with regular polymorphism, as in Java generics.

4.3 Haskell implementation

In this section we will explain in more depth some of the Haskell implementation details, to give a reader with experience in Haskell an overview of the structure of the model. Where we gave an in depth overview of the musical aspects of the model in the previous section, we will here concentrate on a small but representative subset of implementation issues and elide most of the musical details. Hence, to comprehend the matters discussed in this section some Haskell knowledge is required. The HARMTRACE model has become an extensive model and to keep things simple we start with simple code examples and highlight how we solved problems that we encountered when we increased model complexity.

As explained above, the general idea is that we convert an input sequence of chord labels, such as `"C:maj F:maj D:7 G:7 C:maj"` (also shown in Figure 4.1), into a value of a Haskell datatype which captures the function of chords within the sequence. Since we want to abstract from specific keys, we first translate every chord label into a scale degree. For instance, our previous example is in C major, so it translates to `"1:maj 4:maj 2:dom 5:dom 1:maj"`.

4.3.1 Naive approach

Using standard algebraic datatypes, we can mimic the specification in Section 4.2.1 by encoding alternatives as constructors, sequences as arguments to a constructor, and repetitions as lists. A first (and very simplified) approach could be the following:

```
data Piece  = Piece [Func]
data Func   = PT    Ton    | PD    Dom
data Ton    = T_IMaj  Degree
data Dom    = D_VMaj  Degree | D_SD,D SDom Dom
data SDom   = S_IVMaj Degree
```

A piece is represented as a list of functional categories, which can either be tonics or dominants. A tonic is simply the first scale degree, while a dominant might branch into a subdominant and a dominant, or simply be the fifth degree, which is very similar to what we have seen in Section 4.2.1. Finally, all datatypes mount out in a scale degree datatype, which are the leaves of our analysis tree. A scale degree data type consists of a root degree (an integer between 1 and 12) together with a chord class:

```
data Degree = Deg Int Class
data Class  = Maj | Min | Dom | Dim
```

The chord class is used to group chords into a small number of categories based on their internal interval structure. All major chords are grouped under *Maj*, *Min* groups all minor chords, *Dom* groups dominant seventh chords and *Dim* groups the diminished seventh chords, exactly as defined on page 65. We can now encode harmonic sequences as values of type Piece:

$$
\begin{aligned}
&goodPiece, badPiece :: \mathsf{Piece} \\
&goodPiece = Piece\,[PT\,(T_{IMaj}\,(Deg\,1\,Maj))] \\
&badPiece \;\; = Piece\,[PT\,(T_{IMaj}\,(Deg\,2\,Maj))]
\end{aligned}
$$

The problem with this representation is evident: non-sensical sequences such as *badPiece* are allowed by the type-checker. We know that a Tonic can never be a second scale degree: it must be a first scale degree. However, since we do not constrain the Degree argument in T_{IMaj}, we have to make sure at the value-level that we only accept $Deg\,1\,Maj$ as an argument. To guarantee that the model never deals with invalid sequences we would need a separate proof of code correctness.

4.3.2 More type information

Fortunately, we can make our model *more typed* simply by encoding degrees and chord classes at the type level:

$$
\begin{aligned}
&\textbf{data}\ \mathsf{Ton} \;\;\;\; = T_{IMaj}\ \ (\mathsf{Degree\ I}\ \ \mathsf{Maj}) \\
&\textbf{data}\ \mathsf{Dom} \;\;\;= D_{VMaj}\ \ (\mathsf{Degree\ V}\ \ \mathsf{Maj}) \mid D_{SD,D}\ \mathsf{SDom\ Dom} \\
&\textbf{data}\ \mathsf{SDom} = S_{IVMaj}\ (\mathsf{Degree\ IV\ Maj}) \\
&\textbf{data}\ \mathsf{Degree}\ \delta\ \gamma = Deg\ \mathsf{Int\ Class}
\end{aligned}
$$

Now we detail precisely the root and class of the scale degree expected by each of the constructors. We need type-level scale degrees and classes to use as arguments to the new Degree type:

$$
\begin{aligned}
&\textbf{data}\ \mathsf{I}; \textbf{data}\ \mathsf{II}; \textbf{data}\ \mathsf{III}; \textbf{data}\ \mathsf{IV}; \textbf{data}\ \mathsf{V}; \textbf{data}\ \mathsf{VI}; \textbf{data}\ \mathsf{VII}; \\
&\textbf{data}\ \mathsf{Maj}; \textbf{data}\ \mathsf{Dom}; \ldots
\end{aligned}
$$

It only remains to guarantee that Degrees are built correctly. An easy way to achieve this is to have a type class mediating type-to-value conversions, and a function to build degrees:

$$
\begin{aligned}
&\textbf{class}\ \ \ \ \mathsf{ToDegree}\ \delta\ \textbf{where}\ toDegree :: \delta \to \mathsf{Int} \\
&\textbf{instance}\ \mathsf{ToDegree\ I}\ \textbf{where}\ toDegree\ \_ = 1 \\
&\ldots
\end{aligned}
$$

```
class      ToClass γ    where toClass :: γ → Class
instance ToClass Maj where toClass _ = Maj
...
deg :: (ToDegree δ, ToClass γ) ⇒ δ → γ → Degree δ γ
deg root clss = Deg (toDegree root) (toClass clss)
```

If we also make sure that the constructor *Deg* is not exported, we can be certain that our value-level Degrees correctly reflect their type. Sequences like *badPiece* above are no longer possible, since the term T_{IMaj} (*deg* (⊥ :: II) (⊥ :: Maj)) is not well-typed.

4.3.3 Secondary dominants

So far we have seen how to encode simple harmonic rules and guarantee that well-typed pieces make sense harmonically. However, we also need to encode harmonic rules that account for secondary dominants (Spec. 16 at page 66). According to harmony theory, every scale degree can be preceded by the scale degree of the dominant class a fifth interval up. To encode this notion we need to compute transpositions on scale degrees. Since we encode the degree at the type-level this means we need type-level computations. For this we use Generalised Abstract DataTypes (Peyton Jones et al. 2006, GADTs) and type families (Schrijvers et al. 2008). GADTs allow us to conveniently restrict the chord root and class for certain constructors, while type families perform the necessary transpositions for relative degrees. To support chains of secondary dominants we change the Degree type as follows:

```
data Degree_n δ γ η where
  Base_Deg :: Degree_Final δ γ → Degree_n δ γ (Su η)
  Cons_V   :: Degree_n (V/ δ) Dom η → Degree_Final δ γ
           → Degree_n δ γ (Su η)
data Degree_Final δ γ = Deg Int Class
```

We now have two constructors for $\mathsf{Degree_n}$: $Base_{Deg}$ and $Cons_V$. $Base_{Deg}$ is the base case, which stores a Degree and a Class as before. In $Cons_V$ we encode the secondary dominants. Its type specifies that we can produce a $\mathsf{Degree_n}$ for any scale degree δ and class γ by having a $\mathsf{Degree_{Final}}$ for that root and class preceded by a $\mathsf{Degree_n}$ of root $\mathsf{V}/\,\delta$ of the dominant class. The type family V/ transposes its argument degree a fifth up:

```
type family    V/ δ
type instance V/ I   = V
type instance V/ II  = VI
type instance V/ III = VII
...
```

To avoid infinite recursion in the parser we use a type-level natural number in $\mathsf{Degree_n}$. This parameter also serves to control the number of allowed secondary dominants:

```
data Su η
data Ze
type Degree δ γ = Degree_n δ γ (Su (Su (Su (Su Ze))))
```

Typically we use values between 5 and 7 for η. Its value greatly affects compilation time. Other recursive datatypes of which the number of recursive applications must controlled, such as minor fifth insertion (Spec 17) and diminished seventh chord transposition (Spec. 21), are handled in a similar way.

A convenient side-effect of the use of type families is that it gives us fine grained control over the cases we want the secondary dominant specification to apply. For example, in the simple sequence, $[Deg\ 5\ Dom, Deg\ 1\ Maj]$, we prefer to parse the $(Deg\ 5\ Dom)$ as a regular dominant (D_{VMaj}) and not as a secondary dominant ($Cons_V$). For this we use a fictionary degree which we make sure that it will never be parsed, Imp:

```
  -- Impossible to parse
data Imp
instance ToDegree Imp where toDegree _ = -1
type family    V/δ
type instance V/I   = Imp
type instance V/II  = VI
type instance V/III = VII
...
```

4.3.4 Generic parsing

We have seen how to put Haskell's advanced type system features to good use in the definition of a model of tonal harmony. In this section we further exploit the advantages of a well-typed model while defining a generic parser from labelled scale degrees (e.g. `"I:maj IV:maj II:7 V:7 I:maj"`) to our datatype. Nevertheless the same techniques can also be used for other operations on the model, like pretty-printing and comparing datatypes (Magalhães and De Haas 2011).

From the high-level musical structure (e.g. the Ton and Dom datatypes of Section 4.3.2) we can easily build a parser in applicative style mimicking the structure of the types:

```
data Parser α   -- abstract
```

class Parse α **where** *parse* :: Parser α
instance Parse Ton **where**
  parse = T_{IMaj} <$> *parse*
instance Parse Dom **where**
  parse = D_{VMaj} <$> *parse*
    <|> $D_{SD,D}$ <$> *parse* <*> *parse*

For the purposes of this paper we keep Parser abstract; in our implementation we use the `uu-parsinglib` (Swierstra 2009), because our grammar is ambiguous and we can put error-correction to good use, as we explain in Section 4.2.2.

The instances of Parse for Ton and Dom are trivial because they follow directly from the structure of the datatypes. They can even be obtained by syntactic manipulation of the datatype declaration: replace | by <|>, add <$> after the constructor name, separate constructor arguments by <*> and replace each argument by *parse*. The code is tedious to write, and since we have several similar datatypes it becomes repetitive and long.

Because different musical styles can have different harmony rules (e.g. baroque harmony versus jazz harmony) we want our model to be flexible and easy to change or even support multiple models. We solve these problems by *not* writing instances like the one above. Instead, we use *datatype-generic programming* to derive a parser automatically in a type-safe way. Given a collection of datatypes a parser identical to the code above is automatically generated. We use the `instant-generics`, which implements a library similar to that initially described by Chakravarty et al. (2009). Explaining how generic programming works is beyond the scope of this thesis, but our generic parser is entirely trivial. The order of the constructors and their arguments determines the order of the alternatives and sequences; in particular, we avoid left-recursion in our datatypes, since we do not implement a grammar analysis like Devriese and Piessens (2011).

4.3.5 Adhoc parsers

The only truly non-generic parser is that for $\mathsf{Degree}_{\mathsf{Final}}$, which is also the only parser that consumes any input. It uses the type classes ToDegree and ToClass as described in Section 4.3.2. Unfortunately, we are also forced to write the parser instances for GADTs such as $\mathsf{Degree}_{\mathsf{n}}$, since the Instant Generics library does not support GADTs. Although the code trivially mimics the datatype specifications, the instance heads become more complicated, since they have to reflect the type equalities introduced by the GADT. As an example, we show the parser code for $\mathsf{Degree}_{\mathsf{n}}$:

```
instance ( Parse (Degree_Final δ γ)
         , Parse (Degree_n (V/ δ) Dom η      ))
         ⇒ Parse (Degree_n δ      γ     (Su η)) where
  parse =  Base_Deg <$> parse
       <|> Cons_V   <$> parse <*> parse
```

The context of the instance reflects the type of the constructors of $\mathsf{Degree_n}$: $Base_{Deg}$ introduces the $\mathsf{Parse}\ (\mathsf{Degree_{Final}}\ \delta\ \gamma)$ constraint, whereas $Cons_V$ requires $\mathsf{Parse}\ (\mathsf{Degree_n}\ (\mathsf{V}/\ \delta)\ \mathsf{Dom}\ \eta)$ too.

The need for type-level natural numbers becomes evident here; the instance above is *undecidable* for the compiler, meaning that the rules for instance contexts become relaxed. Normally there are constraints on what can be written in the context to guarantee termination of type-checking. Undecidable instances lift these restrictions, with the side-effect that type-checking becomes undecidable in general. However, we are certain that type-checking will always terminate since we recursively decrease the type-level natural number η. This means we also need a *base case instance* where we use the *empty* parser which always fails; this is acceptable because it is never used.

```
instance Parse (Degree_n δ γ Ze) where parse = empty
```

Note how useful the type class resolution mechanism becomes: it recursively builds a parser for all possible alternatives, driven by the type argument η. This also means potentially long type-checking times; fortunately our current implementation remains compilable under a minute.

The code presented in this subsection demonstrates how we solve the main implementation issues we encountered when developing HARMTRACE. The current version of HARMTRACE uses all the techniques described here, but to represent the complete model exhibited in the previous sections, some datatypes and functions had become more complicated. For instance, the current version of HARMTRACE uses GADTs to index all datatypes with the mode of the current key. Using the a new version of the Instant Generics library, which features some preliminary support for GADTs (Magalhães and Jeuring 2011), we can still maintain the same flexibility as with regular algebraic datatypes. In Appendix 7.2 we have printed the complete Haskell code of the module that corresponds to the model as explained in Chapter 5.

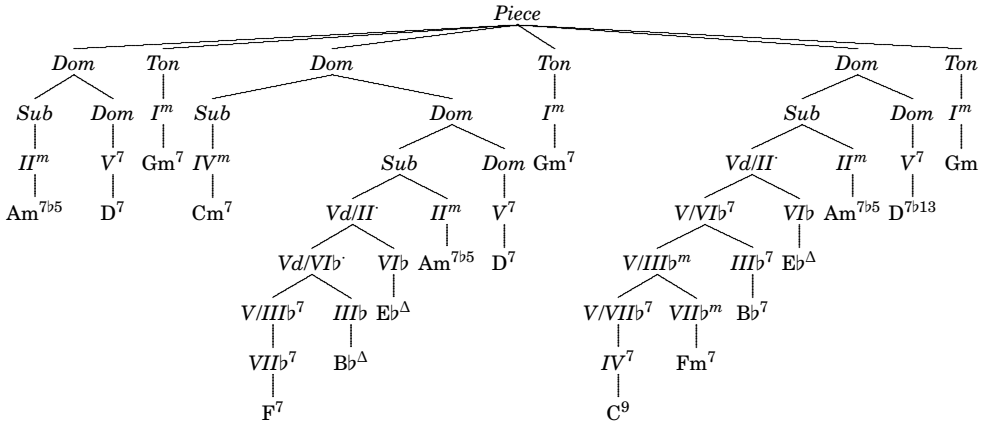


Figure 4.3: The harmony analysis of the B-part of the jazz standard *Autumn Leaves* as derived by HARMTRACE. For convenience we left out the mode subscript Min and we printed the chord labels as is common in Real Book lead-sheets. The key of the piece is G minor.

4.4 Example analyses

In this section we demonstrate the analytic power of the HARMTRACE system. The input presented to HARMTRACE consists of a sequence of plain text chord labels in the syntax defined by Harte et al. (2005), and the output consists of a parse tree similar to those often used in natural language processing. In these trees the input chord labels are the leaves, which are subsequently grouped into scale degrees, scale degree transformations, functional categories, and finally collected in a *Piece* node, as prescribed by the rules of the model. The notation used in the parse trees is identical to the notation used to describe the datatype specifications in the previous section.

We start by analysing the B-part of the *Autumn Leaves* jazz standard as found in the *Real Book* (Various authors 2005). The parse tree of this piece in the key of G minor is depicted in Figure 4.3. Within the piece, the three A^0 D^7 Gm sequences resolve into subdominant, dominant, and tonic categories (Spec. 9, 6, and 5, respectively), forming ii-V-progressions, towards the tonic of the piece. The second and third *Dom* branches of *Piece* display different types of descending fifth movements, which build up tension towards a D^7: the preparation of the B♭ by an F is labelled as a secondary dominant or as a diatonic descending fifth, depending on whether the chord is dominant or major (Spec. 16 or 19).

Although the model has a bias towards a jazz repertoire, it can be used to analyse Bach chorales as well. In Figure 4.4 we present the HARMTRACE analysis of the first 9

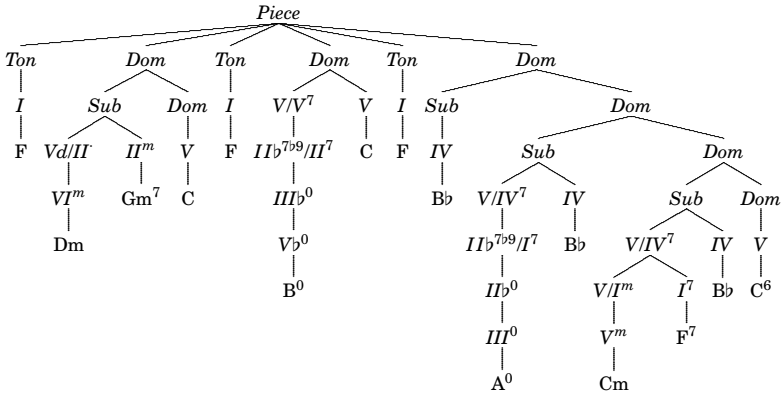


Figure 4.4: The automatic analysis of the first two phrases of J.S. Bach's *Ach Herre Gott, mich treibt die Not*, BWV 349, in the key of F.

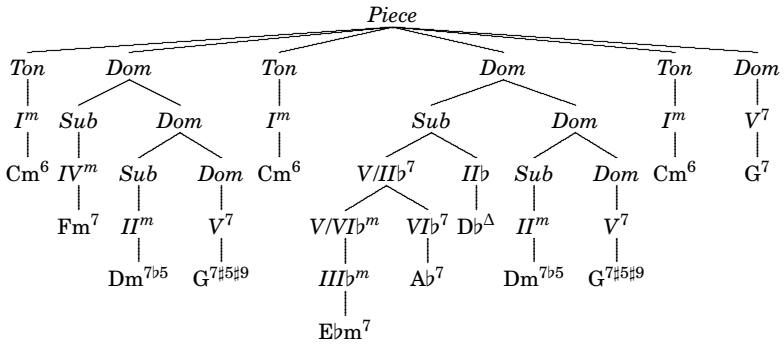


Figure 4.5: An analysis of *Blue Bossa* in C minor.

measures of Bach's *Ach Herre Gott, mich treibt die Not* chorale. The key of the piece is F major. After an introduction of the tonic, a diatonic chain of fifths prepares the dominant, C, which subsequently resolves to the tonic. The next branch prepares a C with a B^0, which is *VII/V*. As explained in the previous section, the B^0 is enharmonic equivalent to $A♭^0$ ($III♭^0$) which is very similar to a $G^{7♭9}$ (denoted by $II♭^{7♭9}/II$, Spec. 22), which is in turn the *V/V* of C. The final branch creates some harmonic movement around the B♭ by preparing the second B♭ with a *VII/IV*. The *VII/X* derivation is identical to the one explained above. The fragment is concluded with a descending fifth preparation of the subdominant that is followed by the dominant.

In Figure 4.5 we show the analysis of another well-known standard, *Blue Bossa*. The progression starts by introducing the tonic, Cm, followed by a perfect cadence. The

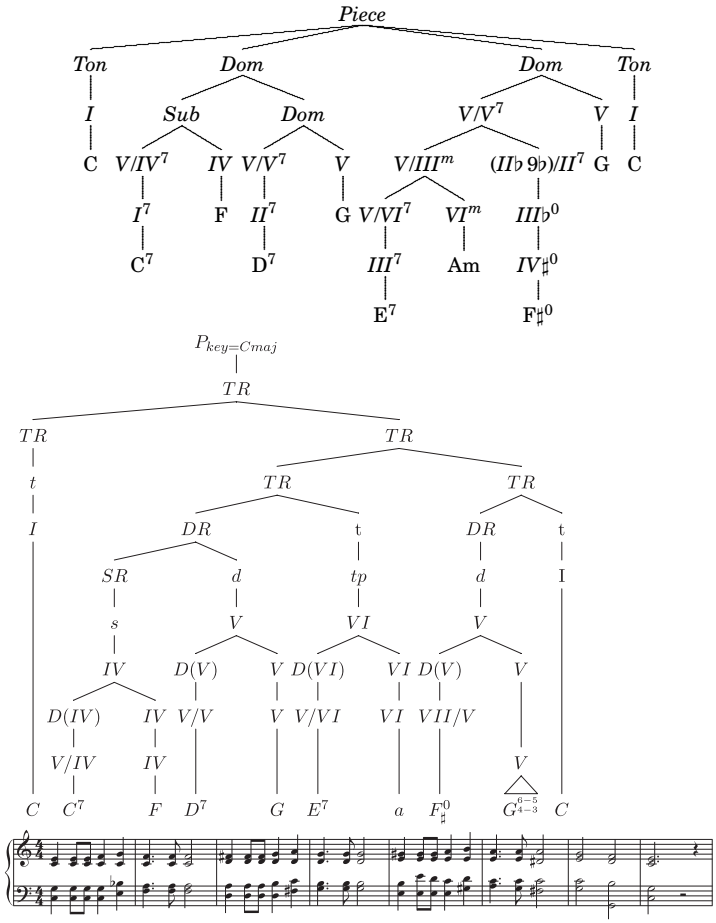


Figure 4.6: Two analyses of a phrase of Bornianski's piece *Tebe Poëm*. The analysis above the score is adopted from Rohrmeier (2011), and the analysis on top is the analysis output by HARMTRACE.

B-part displayed in the second *Dom* branch shows a ii-V-motion to the Neapolitan *II♭* (Spec. 25) and is followed by a ii-V-I to Cm.

Figure 4.6 displays the score and two analyses of Bortnianski's piece *Tebe Poëm*. The analysis on the left is the theoretical analysis proposed by Rohrmeier (2011). Although the notation used by Rohrmeier differs slightly from the notation used in this chapter, the analyses are clearly similar. However, there are also some differences. Rohrmeier connects the tonic, dominant, and subdominant nodes in tonal regions while

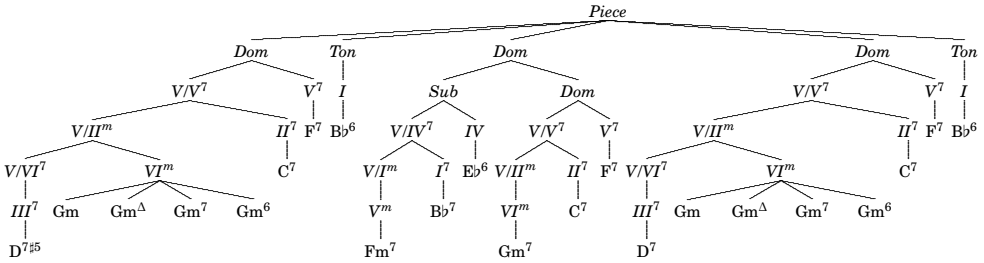


Figure 4.7: An excerpt of the analysis of *It don't mean a thing (if it ain't got that swing)*.

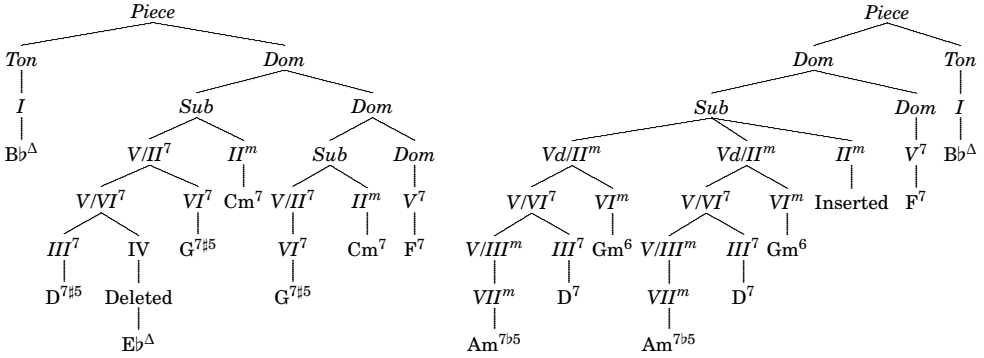


Figure 4.8: Two examples that illustrate error correction. On the left an excerpt of the jazz standard *There is no greater love* and on the right an excerpt of the jazz standard *Someday my prince will come* is displayed.

HARMTRACE does not (we elaborate on this issue in the discussion section). Another difference in derivation is that because we treat the $F\sharp^0$ ($\approx D^{7\flat9}$) as a V/V, the E^7 and Am are analysed as being part of a larger chain of fifths.

In Figure 4.7 we show the HARMTRACE analysis of the jazz standard *It don't mean a thing (if it ain't got that swing)*. The analysis shows how similar Gm chords are grouped under one VI^m node. It furthermore illustrates how the *Sub* and *Dom* nodes are prepared by chains of secondary dominants.

We conclude this section with two small examples that contain error corrections. The example on the left in Figure 4.8 is an excerpt of the jazz standard *There is no greater love*, and the example on the right is an excerpt of the jazz standard *Someday my prince will come*. In the left example the model cannot explain the $E\flat^\Delta$ at that position. Because the D^7 can immediately resolve to the G^7, the parser deletes the $E\flat^\Delta$. The model specification does not allow a *Sub* to translate into a VI^m scale degree. Adding

such a specification would cause a large number of ambiguous solutions, if the diatonic fifth specification (Spec. 19) is not constrained. Therefore, in the example in Figure 4.8 on the right, the model needs a diatonic chain of fifths to explain the VI^m and the parser solves this by inserting a II^m. Corrections like the ones in Figure 4.8 represent typical examples of error corrections in HARMTRACE.

4.5 Experimental results

To demonstrate that the HARMTRACE system can be efficiently and effectively used in practice, we evaluate its parsing performance on two chord sequence datasets: a small dataset that we have used before in Section 3.4, which we will refer to as `small`, and a large dataset that was used in Section 2.3, which we will refer to as `large`. The `small` dataset contains 72 chord sequences that describe mainly jazz pieces. The `large` dataset contains 5028 chord sequences that describe jazz, latin, pop pieces, and a few classical works.

The `small` dataset contains a selection of pieces that were checked manually and "harmonically make sense", while the `large` dataset includes many songs that are harmonically atypical. This is because the files are user-generated, contain peculiar and unfinished pieces, wrong key assignments, and other errors; it can therefore be considered "real life" data. Also, the `large` dataset contains pieces that modulate, and even some pieces that might be considered atonal, e.g. *Giant Steps*. We deliberately chose to use a "real life" dataset to illustrate that HARMTRACE is robust against noisy data, offers good performance in terms of parsing speed, and still delivers analyses that make sense.

4.5.1 Parsing results

When parsing the data we measure the number of parsed chords, deleted chords, inserted chords, and parsing time. These numbers are summarised in Table 4.1. HARMTRACE is compiled using GHC version 7.0.3., and both runs were performed on the same Intel Core 2 6600 machine running at 2.4 GHz with 3 GB of random access memory.

On the `small` dataset the HARMTRACE model performs very well. The songs are parsed quickly and on average fewer than one chord per song is deleted. Also, fewer than three insertions are necessary for a piece to parse, on average. It would have been possible to adapt the model in such way that the `small` dataset would parse without any errors, as was done in Chapter 3. However, we chose to accept this small number of error corrections and keep our grammar small and easy to comprehend. The dataset

Dataset	del/song	ins/song	cor/song	chords/song	time/song	tot. time
`small`	0.83	2.79	3.63	42.49	10.00 ms	0.72 s
`large`	3.38	9.85	13.24	62.05	76.53 ms	384.81 s

Table 4.1: The deleted, inserted, and total number of corrections per song; the total number of chords per song; the parsing time per song; and the total parsing time in seconds.

is parsed within a second.

For the `large` dataset the parsing time per song increases considerably, mostly because the ambiguity of our model can make the error-correction process rather expensive. However, the 5028 chord sequences are still parsed reasonably fast, in 6 min 24 s. The number of error corrections increases considerably, but the parser never crashes or refuses to produce valid output. The higher number of error corrections is expected, since this dataset contains songs with modulations, atonal harmonies, and a variety of errors. Still, HARMTRACE keeps the number of deleted chords under six percent of the total chords parsed.

When we compare the parsing results of the `small` dataset with the results of Section 3.4, we notice that HARMTRACE is much faster. The Java-based parser uses more than 9 min to parse the dataset. We cannot compare the parsing results of the `large` dataset because the majority of the pieces is rejected by their grammar. This emphasises how important the error-correction process is. Nevertheless, HARMTRACE parses the `large` dataset quicker than the java-based parser parses the `small` dataset.

4.6 Discussion

We have presented HARMTRACE, a system that automatically analyses sequences of musical chord labels. Implementing our system in Haskell has proven to be a profitable decision, given the advantages of error-correcting parsers and datatype-generic programming. We have shown that HARMTRACE can handle corpora of considerable size, parses chord progressions fast, and is robust against noisy data. Consequently, we solved the most problematic parsing issues that were raised in Chapter 3. However, HARMTRACE currently features only parallel key distinction, and no support for full modulation. Although the model presented has a bias towards jazz harmony, we have furthermore shown that it can be used to analyse some classical works as well.

If we compare HARMTRACE to other models of tonal harmony we notice various differences. The theoretical work of Steedman (1984), for instance, focuses only on the

structure of the (very particular) style of 12-bar blues, whereas our model aims to formalise the core of tonal harmony with a bias towards jazz harmony, including the 12-bar blues. Although our work draws on the work of Rohrmeier (2007, 2011), there are also considerable contrasts. The most pronounced difference to Rohrmeier's CFG is that the latter features modulation/tonicisation. By tonicising to a local tonic, chords are analysed with respect to that local tonic. As a consequence, his approach can explain secondary dominants by tonicisation, while the HARMTRACE model uses a more jazz-oriented approach to harmony by identifying ii-V-I motions, e.g. Figure 4.3 and Figure 4.5. For instance, in a progression in the key of C major, after moving to the subdominant, F can be viewed as a local tonic allowing the derivation of Gm and C as local subdominant and local dominant. A benefit of Rohrmeier's approach is that it is also possible to derive B♭ C as local subdominant/dominant pair. A specification for this would be easy to add to the HARMTRACE model. However, implementing both tonicisations and secondary dominants, as Rohrmeier suggests, will be problematic since both rules explain the same phenomena. After all, the preparation F by C can be explained both by the tonicisation rules as well as by the secondary dominant rules. This will inevitably lead to an explosion of ambiguous solutions for a harmony progression featuring secondary dominants.

Another difference is that Rohrmeier groups tonics and dominants into higher order phrases or functional regions. He acknowledges that these rules are highly ambiguous, but chooses to keep them for theoretical completeness. The problem is that the harmonic information alone generally does not provide enough information for determining phrase boundaries. For instance, it is unclear whether *Ton Dom Ton* represents a half cadence phrase followed by a tonic (*Ton Dom*) (*Ton*), or an introduction of the tonic followed by a perfect cadence phrase (*Ton*) (*Dom Ton*). We believe that such clusterings should be done in a post-processing step, based on metrical positions and phrase length constraints.

On the whole, when we compare HARMTRACE to other models of tonal harmony, we observe that most models remain purely theoretical. This is regrettable because although theoretical work can yield valuable insights, having a model that is implementable allows it to be evaluated empirically and used in practice. Hence, we argue that if a model designer wants their model to have practical value, they should keep in mind how the model can be implemented. As we have seen in this paper, it may take state-of-the-art programming techniques to create a model that is maintainable, has expressive power, and yet remains fast. We are confident that HARMTRACE will contribute to new insights in the modelling of harmonic analysis. Moreover, HARMTRACE will prove itself useful in tasks such as harmonic similarity estimation and chord labelling, as we will see in the next two chapters of this thesis.

Chapter 5

Context-aware harmonic similarity

WITH the rapid expansion of digital repositories of music, such as iTunes, Spotify, last.fm, and the like, efficient methods to provide content-based access to this kind of music repositories have become increasingly important. To be able to cluster documents, a notion of the similarity between these documents is essential. Hence, within Music Information Retrieval (MIR), the development of musical similarity measures plays a prominent role. Music can be related in many different aspects, e.g. melody, genre, rhythm, etc.; this chapter focuses on similarity of musical harmony. Retrieving music based on its harmony structure has many useful purposes. For instance, when a user is interested in finding cover-songs, or songs of a particular harmonic family, e.g. blues harmonies, one should relate pieces harmonically. Also for classifying variations over a theme in baroque music searching on the basis the harmonic similarity is desirable.

To be able to understand why two chord sequences are harmonically related, we believe it is important to examine chords not only in isolation but also the *context* in which they occur. For this, we draw greatly on classical and jazz harmony theory. In the last decades, many music theorists have studied tonal harmony and observed that within a sequence not every chord is equally important. This suggests that tonal harmony is organised hierarchically. Within a sequence of chords, some chords can be removed leaving the global harmony structure intact, while removing other chords can significantly change how the chord sequence is perceived.

Nowadays there is a rich body of literature that aims to explain the order and regularities in Western tonal harmony, and various ways to analyse the *function* of a chord in its tonal context have been proposed (Riemann 1893; Schenker 1935; Lerdahl and Jackendoff 1996). Unfortunately, the majority of these theories are formulated rather informally and lack descriptions with mathematical precision or computational executability. Although there are exceptions, like the Tonal Pitch Space model (Lerdahl 2001, see also Chapter 2) and David Temperley's Melisma (Temperley 2001), the

lack of mathematical precision has hampered the successful application of harmony models to practical MIR related tasks, such as automatic analysis, similarity estimation, content-based retrieval, or the improvement of low-level feature extraction. In Chapter 3 we presented a proof-of-concept in which we connected automatic harmonic analysis to harmonic similarity measures. In this chapter we will expand on these ideas and demonstrate how we can use the HARMTRACE harmony model, presented in Chapter 4, can help capturing harmonic similarity estimation.

Contribution. In this chapter we present four novel harmonic similarity measures. Two of these similarity measures are based on common harmonic analysis trees and two similarity measures are based on local alignment. Three of these similarity measures depend on the HARMTRACE harmony model up to a considerable extent and use automatic harmonic analyses for determining or improving the harmonic similarity estimation. Furthermore, we present a new adaptation of the Gupta and Nishimura algorithm for finding the largest common embeddable subtrees in $O(nm)$, where n and m are the number of harmonic annotations in the compared sequences. We evaluate all four similarity measures on the large corpus of 5,028 chord sequences presented in Section 2.3.1 and compare the retrieval with the three other harmonic similarity measures introduced in Chapter 2. The results show that a HARMTRACE based alignment solution performs best and we discuss the various musical and computational aspects that play a role in large scale similarity estimation.

The remainder of this chapter is organised as follows. Most of the related work on harmonic similarity has been discussed in Section 1.4.2 and the related work on harmonic models has been reviewed in Section 4.1. Nevertheless, we will briefly recapitulate the most important contributions in Section 5.1. Next, we define two common embeddable subtree based harmonic similarity measures (Section 5.2.5 and Section 5.4) and two similarity measures based on maximum local alignment scores (Section 5.3 and Section 5.3.1). In Section 5.5 we compare the retrieval performance of these new similarity measures to the three harmonic similarity measures presented in Chapter 2. Finally, we conclude this chapter with a discussion on the different properties of the harmonic similarity measures and their effect on retrieval performance in Section 5.6.

5.1 Related work

The estimation of harmonic similarity has been investigated within the MIR community before. However, generally these are embedded into larger retrieval systems and take audio or score information as input, (e.g., Pickens and Crawford 2002) or (Serrà et al. 2008). We believe it is wise to abstract from the feature extraction and

chord transcription preprocessing steps and solely focus on the similarity of the harmonic progression, which can conveniently be represented as a sequence of symbolic chord labels. A benefit of evaluating only a similarity measure is that errors caused by the feature extraction or chord labelling methods do not influence the retrieval evaluation. Strangely, the research purely investigating the similarity of chord sequences has received far less attention. We are aware of only a few other systems that estimate the similarity of symbolic chord labels: the Tonal Pitch Step Distance (TPSD), the Chord Sequence Alignment System (Hanna et al. 2009, CSAS) and the proof-of-concept presented in Chapter 3. The calculation and the comparison of the TPSD and CSAS is covered elaborately in (De Haas et al. 2011c) and in Chapter 2. Nonetheless, we also briefly introduce them here.

The TPSD uses Lerdahl's Tonal Pitch Space (2001, TPS) as its main musical model. TPS is a model of tonality that fits musicological intuitions, correlates well with empirical findings from music cognition, and can be used to calculate a distance between two arbitrary chords. The TPS model takes into account the number of steps on the circle of fifths between the roots of the chords, and the amount of overlap between the chord structures of the two chords and their relation to the global key. The general idea behind the TPSD is to use the TPS to compare the change of perceived chordal distance to the tonic over time. For every chord, the TPS distance to the key of the sequence is calculated, resulting in a step function. Next, the distance between two chord sequences is defined as the minimal area between the two step functions over all possible horizontal circular shifts. To prevent longer sequences from yielding larger distances, the score is normalised by the duration of the shortest song.

The CSAS (Hanna et al. 2009) is based on local alignment: by performing elementary deletion, insertion, and substitution operations, one chord sequence is transformed into the other. The actual similarity value is defined as the total sum of all edit operations at all beat positions. To improve the retrieval performance of the classical alignment approach, Hanna et al. experimented with various musical data representations and substitution functions. They found a key-relative representation, based on the interval between the root of the chord and the key, to work well and preferred substituting only when the chord root and triad were not identical. In the experiments in (De Haas et al. 2011c) the CSAS outperformed the TPSD in 4 of the 6 tasks.

In Chapter 3 we explore a third approach to harmonic similarity. We use a context-free grammar to perform an automatic analysis and analyse the harmonic analysis parse trees by examining the largest common embeddable subtree. While we addressed the automatic harmonic analysis in Chapter 4, we will elaborate on the similarity matching of the harmonic annotations in this chapter. Hence, three of our similarity measures depend on the harmonic analyses provided by the HARMTRACE harmony model. The

HARMTRACE harmony model transports the ideas of Rohrmeier's (2007; 2011) grammar to a functional setting, solving many of the typical problems associated with context free parsing. In the next sections we will propose four new similarity measures. Two similarity measures are based on the comparison of harmonic analysis trees as obtained by the HARMTRACE harmony model and two similarity measures are based on sequence alignment.

5.2 Related subtree based similarity

Trees are omnipresent in computer practice and not surprisingly trees are among the best studied data structures in computer science. Trees are used to represent file systems, XML documents, websites, the grammatical structure of a language, or evolutionary ascendancy, to name only a few of the many practical uses. What is shared among all of these applications is that trees are used to structure and order information. Having a tree data structure that represents certain kind of knowledge, the question emerges how one can compare these trees. In this section we aim to provide some new insights into the similarity between trees by examining which hierarchical structures are shared between two trees. More specifically, following Chapter 3, this chapter proposes similarity estimation based on the *Largest Common Embeddable Subtree* (LCES) problem for two harmonic analysis trees. Given two trees, the aim is to find the largest subtree that can be embedded in these two trees.

5.2.1 Preliminaries

A rooted tree is a structure denoted with $T = (V, E, T_{root})$, where V is a finite set of nodes, E is the set of edges connecting nodes in V, and T_{root} is the root of the tree T. The function $V(T)$ provides access to V in T. The nodes of the parse trees generated by HARMTRACE are all labelled and the label of node v can be retrieved with the function $label(v)$. The subtree of T rooted at node v is denoted with T_v and $child(v)$ returns the subset of V that have v as parent. A single child of a node v is denoted with c^v. The function $post(v)$ returns the post order number that is assigned to a node v in a postorder traversal. $depth(T)$ denotes the depth of a tree, i.e. the number of nodes in the longest path from leaf to the root. Finally, the $degree(T)$ is the degree of a tree, i.e. the maximum number of children.

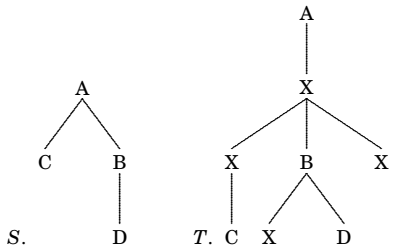


Figure 5.1: An example of a tree S that can be embedded in a tree T preserving the ordering and the labelling.

5.2.2 Tree embeddings

Within the literature different definitions exist about what defines a topological embedding. In this, we adopt the definitions of Kilpeläinen (1992). We say that a tree S is *topologically embeddable* in a tree T if there exists an embedding of S into T. An embedding is an injective function mapping each node in S to a node in T, that preserves ancestorship and optionally the labelling and order. As we explained in Section 1.2, music unfolds over time and we think that the order of an harmonic analysis is important and should be preserved. Hence, we will only consider ordered trees, ordered inclusions and ordered embeddings in this chapter; Figure 5.1 shows an example of an included tree. A Tree S is an ordered embedding of a tree T if there exists a function $f : V(S) \rightarrow V(T)$ and that for all nodes u and v in S holds that:

1. $f(u) = f(v)$ if and only if $u = v$,
2. u is a ancestor of v in S if and only if $f(u)$ is an ancestor of $f(v)$ in T,
3. $label(u) = label(f(u))$.
4. $post(u) < post(v)$ if and only if $post(f(u)) < post(f(v))$.

5.2.3 Largest common embeddable subtrees

A natural extension to the problem of determining whether a tree S can be embedded into a tree T is the *Largest Common Embeddable Subtree* (LCES)[23] problem: given

[23] Strictly speaking, *Largest*, might be considered a sub-optimal term for describing the common embeddable subtree phenomena at hand, because possibly more of these trees may exist. Hence, *Maximum* might be a more appropriate term. However, because the term Largest Common Embeddable Subtree has been used in a considerable number of articles, we will use *Largest* in this article as well.

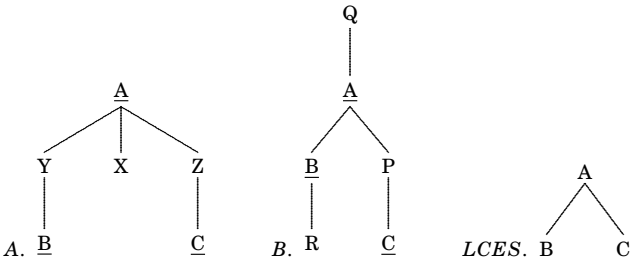


Figure 5.2: Two labelled trees A and B and an ordered Largest Common Embeddable Subtree $LCES$. The matching labels in A and B are underlined.

to trees A and B, determine the largest tree S that can be embedded into both A and B. It is important to observe that the LCES is the maximum mapping $f : V(A) \in V(B)$ between the nodes of A to the nodes of B, respecting the order, the ancestorship relations, and the labels of the nodes. To get some insights into the mapping f this mapping can be visualised as a tree as in Figure 5.2.

5.2.4 An algorithm for finding the LCES

In Chapter 3 we showed that the size and depth of the labelled LCES can be used to estimate the similarity of two trees. However, the algorithm presented in Section 3.3.2 is unable to deal with duplicate labels. This problem has been solved by an relabelling traversal, but this forces the first occurrence in the one tree to match always with the first occurrence of this node in the other tree. Gupta and Nishimura (1995, 1998) propose algorithms for finding the LCES for both the ordered and the unordered unlabelled trees. In order to not having to relabel all nodes, we present in this section an adaptation of the Gupta and Nishimura algorithm for ordered trees that takes the labelling of the nodes into account and is robust against duplicate labels. Both our algorithm as well as the algorithms of Gupta and Nishimura are based on the subtree isomorphism algorithm of Matula (1978). In all solutions the LCES problem is solved with dynamic programming. Explaining the well-known dynamic programming paradigm is beyond the scope of this thesis, but for a fine introduction we refer to (Cormen et al. 2001, Chap. 15); also Matula (1978) explains dynamic programming in the context of subtree isomorphism. The general idea is to construct the matching f by traversing two trees A and B in post order and recursively calculate and store the size of the maximum matching for each pair of subtrees A_u and B_v and respect the ancestor relations, order, and labels of the nodes. Observe that a maximum matching can be found by recursively calculating the maximum matching of the following combinations of subtrees: if c^u and c^v denote arbitrary children of u and v, respectively, then for each

pair of subtrees A_u and B_v there exist a common embeddable subtree if:

1. there exists at least one common embedding into A_{c^u} and B_{c^v}, or
2. there exists at least one common embedding into A_{c^u} and B_v, or
3. there exists at least one common embedding into A_u and B_{c^v}, or
4. $label(u) = label(v)$.

The first case captures the matching between the subtrees rooted children of u and the subtrees rooted children of v. Such a maximum matching can be found by maximising the matching between the children of v and u because the children c^u and c^v have been matched before (assuming both trees are traversed from the leaves up). The maximum is found by constructing and matching a weighted bipartite graph $G = (child(v), child(u), S)$, where S contains the edges between $child(v)$ and $child(u)$. The weights of the edges correspond to the size of the matching between the subtrees rooted by the two matched children. In the second case we can exploit the property that, if there exist a matching between the subtree A_{c^u} of A_u and B_v, this matching is also contained in its super tree A_u and B_v. In this cases the root of the tree A_u is skipped. The third case captures the same phenomenon for the subtrees A_u and B_{c^v}. Finally, the fourth case is trivial: if the labels of v and u match, one can create a tree consisting of one node u with $label(u) = label(v)$ this tree will always be embeddable into A_u and B_v. Note that the fourth case can be true in conjunction with the previous three cases.

We present a function GETLCES that calculates the LCES using an integer array L; in $L[i,j]$ we store the size of the LCES that is embeddable into both A_u and B_v, where $i = po(A_u)$ and $j = po(B_v)$. We fill the entries of L by processing A and B in post order. The advantage of using an array L is that when two arbitrary internal nodes u and v are matched, the sizes of the LCESs of their children c^u and c^v do not have to be calculated but can be looked up in $L[po(c^u), po(c^v)]$ where $c^u \in child(u)$ and $c^v \in child(v)$. If both nodes u of A and v of B are leafs and their labels match we store a 1 and a 0 otherwise. If at least u or v is an internal node then the size of the LCES

embeddable into A_u and B_v is given by the following recursive formula:

$$\begin{aligned} L[u,v] &= x+y \\ x &= \max \begin{cases} \text{WBMATCH}(child(u), child(v)) \\ \max\{A[u,c] \mid c \in child(u)\} \\ \max\{A[c,v] \mid c \in child(v)\} \end{cases} \\ y &= \begin{cases} 1 \text{ if } label(u) = label(v) \\ 0 \text{ otherwise} \end{cases} \end{aligned} \tag{5.1}$$

Note that in the recursive formula above, WBMATCH refers to a function that calculates a maximum weighted bipartite matching. Because we only consider ordered trees, we are solely interested in ordered weighted matchings, i.e. the edges in the bipartite graph are not allowed to cross, and this makes this problem easier and solvable with another dynamic programming approach.

We present a new algorithm for finding a maximal ordered matching. Given two subtrees A_u and B_v we construct a bipartite graph $G = (child(A_u), child(B_v), S)$, where S will become the set of edges connecting $V(A_u)$ and $V(B_v)$. Recall that to respect the ordering the edges in S cannot cross. If S is a maximum matching between $child(A_u)$ and $child(B_v)$, the matchings between prefixes of $child(A_u)$ and $child(B_v)$ contained in S are also maximal, this suggests another a dynamic programming solution. To ensure that the matching S is maximal, we construct an array M which stores the size of the matching so far. M is filled by calculating the recurrence 5.2 for every combination of nodes in $child(A_u)$ and $child(B_v)$ in a standard dynamic programming procedure. In Equation Equation 5.2 i and j correspond to the indices of the children in $child(A_u)$ and $child(B_v)$, respectively.

$$
\begin{aligned}
M[i,j] \;=\; & \begin{cases} q & \text{if } i=1 \wedge j=1, \\ \max(q, M[1,j-1]) & \text{if } i=1, \\ \max(q, M[i-1,1]) & \text{if } j=1, \\ p & \text{otherwise} \end{cases} \\
p \;=\; & \max \begin{cases} M[i,j-1]+q, \\ M[i-1,j]+q, \\ M[i-1,j-1]+q \end{cases} \\
q \;=\; & \begin{cases} 1 \text{ if } label(u_i) = label(v_j) \;\wedge \\ \qquad u_i \text{ has not been matched before} \;\wedge \\ \qquad v_j \text{ has not been matched before,} \\ 0 \text{ otherwise} \end{cases}
\end{aligned}
\tag{5.2}
$$

It is not difficult to see that the calculation of M depends on the pairwise comparisons between the nodes in $child(A_u)$ and $child(B_v)$. To fill the table M, every node in $child(A_u)$ is compared exactly ones with every node in $child(B_v)$. This yields nm comparisons and a $O(nm)$ time complexity, where n and m are the nodes in $child(A_u)$ and $child(B_v)$, respectively.

If, in the calculation of L and M in WBMATCH and GETLCES, we keeps track of the set matching edges K, matching the nodes in A and B, the hierarchical structure of the LCES can be reconstructed by filtering out the nodes in A or B that are not contained in a match in K. The running time of GETLCES is dominated by the pairwise combination of all nodes in A and B. If the degree of the nodes is assumed to be constant, $size(A) = n$ and $size(B) = m$, the running time of the total algorithm is:

$$
\begin{aligned}
O\left(\sum_{v \in V(A)} \sum_{u \in V(B)} (|\, degree(v) \,| \cdot |\, degree(u) \,|) \right) \;&= \\
O\left(\sum_{v \in V(A)} |\, degree(v) \,| \cdot \sum_{u \in V(B)} |\, degree(u) \,| \right) \;&= \\
O(|\, V(A) \,| \cdot |\, V(B) \,|) \;&= O(nm)
\end{aligned}
\tag{5.3}
$$

5.2.5 Analysing the LCES

Having an ordered labelled LCES, we can analyse some of its properties in the same manner as we did in Chapter 3. Since GETLCES maximises the size of the LCES, using the size is our first pick, but also the depth could be used. However, estimating

the grammaticality of the LCES by counting the edges that correspond to a context-free rule is more problematic. Although in principle the datatype specification as specified in Section 4.2.1 can be translated into a context-free grammar, this would require some very complicated and difficult to write generic function that restrained which would have required a significant time investment. Hence, we decided to first investigate the performance of a size-based LCES similarity measure.

Because the size of the LCES yield the best results after the grammar violation distance presented in Section 3.3.3, we present the following similarity measure:

$$\text{LCESSIZE}(A,B) = \frac{L[n,m]}{n} \cdot \frac{L[n,m]}{m} \tag{5.4}$$

In Equation 5.4 L is obtained by the function GETLCES and recall that $L[n,m]$ stores the size of LCES where n and m are the number of nodes in two harmony trees A and B, which are obtained by parsing the two compared chord sequences with the HARMTRACE harmony model as explained in Section 4.2.1. Equation 5.4 normalises the similarity measure by the size of the trees A and B, and the LCESSIZE returns a value between 0 and 1. We prefer this particular normalisation over, for example, the arithmetic mean of the alignment scores divided by the lengths n and m, because the normalisation in Equation 5.4 will only treat A and B as similar when both the terms $\frac{L[n,m]}{n}$ and $\frac{L[n,m]}{m}$ are close to 1.0. In this sense the normalisation in Equation 5.4 is more strict and less forgiving than, for instance, the arithmetic mean.

5.3 Sequence based similarity

In this section we present a different approach towards harmonic similarity based on local alignment. Local alignment has been used before to estimate both melodic similarity (Van Kranenburg et al. 2009) and harmonic similarity (Hanna et al. 2009). We will demonstrate how we can exploit information about a automatic harmonic analysis by the HARMTRACE harmony model can help quantifying harmonic similarity.

After having obtained an harmonic analysis from the HARMTRACE harmony model, a chord is categorised as being part of either a dominant, sub-dominant, or tonic structure (Spec. 4–11, Page 65). Furthermore, we also annotate whether a chord is part of secondary dominant preparation (Spec. 16–17, Page 66) and label whether it has been transformed (Spec. 20–21, Page 67). We hypothesise that these annotations are helpful in determining harmonic similarity. Hence, we represent an annotated chord as a quintuple of the following form: (X, $\mathbf{c}$, *func*, *prep*, *trans*), where X represents a scale

degree, **c** a chord class (as defined in Section 4.2.1), *func* the functional category, *prep* the functional preparation, e.g. being part of a secondary dominant (V/X), and *trans* a scale degree transformation, e.g. a tritone or diminished seventh substitution. Note that by taking the functional preparations and categories into account, the local tonal context is incorporated into the matching. For estimating the similarity between two sequences of these annotated chords we calculate the alignment score obtained in a classical alignment procedure (Smith and Waterman 1981).

The quality of an alignment also depends on the insertion, deletion, match, and mismatch parameters. We use a constant insertion and deletion penalty of -2 and we define the similarity between the annotated chords as a function, $sim(ci,dj) \in [-1,6]$, that takes a pair of chords, c_i and d_j, and returns an integer denoting the (dis-)similarity. Here i and j denote the beat position of c_i and d_j in the compared chord sequences C and D.

$$
\begin{aligned}
&sim\,((X_1,\mathbf{c}_1,func_1,prep_1,trans_1),(X_2,\mathbf{c}_2,func_2,prep_2,trans_2)) = \\
&\quad \textbf{if}\; X_1 \equiv X_2 \wedge \mathbf{c}_1 \equiv \mathbf{c}_2 \;\textbf{then}\; 2 + m_{prep} + m_{trans} \;\textbf{else}\; -1 \\
&\qquad \textbf{where}\; m_{prep} \quad = sim_{prep}\;(prep_1\;, prep_2\;) \\
&\qquad\qquad\quad m_{trans} \quad = sim_{trans}\,(trans_1, trans_2) \\
&\qquad\qquad\quad sim_{trans}\; a\; b = \textbf{if}\; a \equiv b \;\textbf{then}\; 1 \;\textbf{else}\; 0
\end{aligned}
$$

Within sim, the function $sim_{prep}\,(prep_1, prep_2) \in [0,3]$ compares two possible scale degree preparations, returning 3 is the preparation is identical, 2 if both preparations involve the same fifth jump, 1 if they are both a preparation, and 0 in all other cases.

The final similarity score is obtained by calculating the optimal alignment between two annotated chord sequences and normalising the alignment score. Because the prefix of an optimal alignment is also an optimal alignment, an optimal solution can be found by exploiting the dynamic programming paradigm. To ensure that the alignment is maximal, we construct an array T which stores the cumulative alignment score so far. T is filled by calculating the recurrence below for every combination of annotated chords in the sequence C and D in another dynamic programming procedure.

$$
T[i,j] = \max \begin{cases} T[i,j-1] - 2, \\ T[i-1,j] - 2, \\ T[i-1,j-1] + sim(a_i, b_j), \\ 0 \end{cases} \tag{5.5}
$$

The actual alignment can be obtained by keeping track of the path trough T, starting at $T[n,m]$, where n and m are the sizes of C and D, respectively. The function $selfsim(C)$ returns the maximum score that can be obtained for the chord sequence C

by the alignment procedure: its self similarity. We obtain our final similarity measure, $\text{SIM}(C,D) \in [0,1]$, by normalising the sum of alignment scores, $T[n,m]$, by the sizes of C and D:

$$\text{HARMTRACEALIGN}(C,D) = \frac{T[n,m]}{selfsim(C)} \cdot \frac{T[n,m]}{selfsim(D)} \tag{5.6}$$

5.3.1 Basic chord sequence alignment

To evaluate the effect of the HARMTRACE harmony model on retrieval performance, we compare it to a similar alignment system, named TRIADALIGN. In TRIADALIGN we use the exact same alignment procedure, but the similarity function for individual chords, sim, is replaced by sim_{triad} that does not use any additional model information.

$$sim_{triad}\,((X_1,triad_1),(X_2,triad_2)) = \\ \quad \textbf{if}\, X_1 \equiv X_2 \wedge triad_1 \equiv triad_2\, \textbf{then}\, 4\, \textbf{else}\, -1$$

Here, *triad* denotes only whether the chord is major or minor, and the X represents the scale degree, as defined in the previous sections. Note that the TRIADALIGN system is very similar to the CSAS, but uses slightly different parameters and normalises the alignment score.

5.4 LCES based similarity revisited

As we will see in Section 5.5 the LCESSIZE performs not as good as we initially hoped for. On the other hand, it makes sense that, although the LCESSIZE captures similarity up to a considerable degree, the size property (and most certainly also the depth property) of the LCES might too basic for large scale similarity estimation. After all, the information in the LCES, i.e. the actual harmonic analysis that is shared among two pieces and the duration of the chords, is barely used. Some parts of the harmonic analysis are more important than others in the estimation of harmonic similarity. In Section 5.2.5 we solely used the equality of the labels, but as we have seen in Chapter 4, the HARMTRACE harmony model outputs a well-typed tree structure and the datatypes stored in the nodes can be used for comparison as well. Moreover, in the previous section we already defined functions for comparing functional annotations, scale degree transformation and scale degree preparations and used them in aligning annotated chord sequences, which can be reused form comparing nodes in two parse trees.

We use sim_{prep}, sim_{trans} as defined in the previous section and also for sim_{func} and sim_{chord} the same weighting used. sim_{func} and sim_{chord} are defined as follows:

$$sim_{func}(Func_1, Func_2) = \textbf{if } Func_1 \equiv Func_2 \quad \textbf{then } 1 \textbf{ else } 0$$
$$sim_{chord}(X_1, \mathbf{c}_1, X_2, \mathbf{c}_2) = \textbf{if } X_1 \equiv X_2 \wedge \mathbf{c}_1 \equiv \mathbf{c}_2 \textbf{ then } 2 \textbf{ else } 0$$

Next, we replace y in Equation 5.1 (page 90) with:

$$y = sim_{tree}(u, v) \cdot \min(dur(u), dur(v)) \tag{5.7}$$

The function sim_{tree} is a function that applies the appropriate function, i.e. sim_{func}, sim_{prep}, sim_{trans}, or sim_{chord}, when both u and v are of the same type, e.g. both are a scale degree preparations, and returns 0 otherwise. Another important factor that is discarded in the LCESSIM is the duration of the chords. The function dur returns the duration of a node[24]. The durations of the chords are propagated from the leaves up, where the duration of every internal node is the sum of the durations of its children. As a consequence L, will store the maximum matching between the nodes of two compared harmonic analysis trees weighted by $sim_{tree}(u, v)$. Finally, we obtain our fourth similarity measure by normalising the maximum weight by the self similarity:

$$\text{LCESSIM}(A, B) = \frac{L[n, m]}{selfsim(A)} \cdot \frac{L[n, m]}{selfsim(B)} \tag{5.8}$$

5.5 Evaluation

We compare the retrieval performance of the similarity measures HARMTRACEALIGN, TRIADALIGN, LCESSIZE and LCESSIM, presented in this chapter, with the similarity measures presented in Chapter 2: the TPSD (Section 2.2, page 27), CSAS (Section 2.1.1, page 22), and BASELINE (Section 2.3, page 30) . We pick the best performing TPSD and the CSAS variants that only use the triadic chord information. We perform a very similar experiment as in Section 2.3 and we use the same chord sequence corpus. This corpus consists of 5,028 unique user-generated Band-in-a-Box files that are collected from the Internet. The files consist of user generated chord sequences and are filtered and preprocessed as described in Section 2.3.1.

Within the corpus, 1,775 songs contain two or more similar versions, forming 691

[24] In practice the sim_{tree} and dur functions are implemented using Haskell's instance resolution mechanism

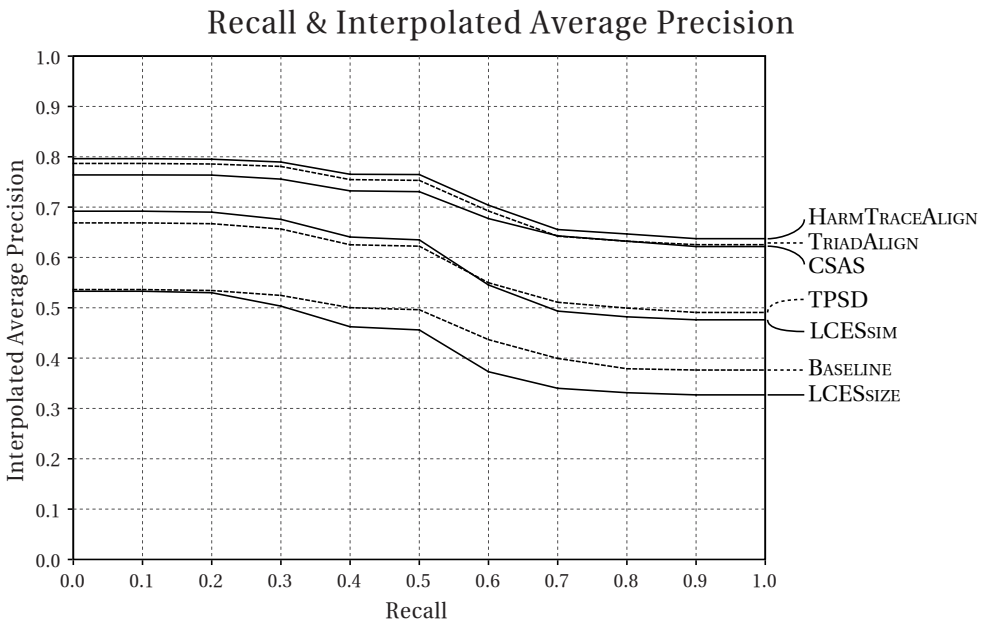


Figure 5.3: A precision recall plot displaying the retrieval performance of all tested similarity measures. Some lines have been dotted to make the results easier to distinguish, it has no additional meaning

classes of songs. Within a song class, songs have the same title and share a similar melody, but may differ in a number of ways. Because multiple chord sequences describe the same song we can set up a *cover-song*-finding experiment. The title of the song is used as ground-truth and the retrieval challenge is to find the other chord sequences representing the same song. Although the dataset was automatically filtered to exclude identical or erroneous pieces, it still includes many songs that are harmonically atypical. The reason for this is that the files are user-generated, and the dataset is to large to manually check all files for inconsistencies. Nevertheless, any *real life* dataset will always contain a certain amount of noise, and our similarity measures should be robust against that.

We analyse the rankings obtained from the compared similarity measures as we have done in Chapter 2 by probing the average interpolated precision at 11 different recall levels and by calculating the Mean Average Precision (MAP). The MAP is the average precision averaged over all queries, and is a single-figure measure between 0 and 1 (Manning et al. 2008, Chap. 8, p. 160). We tested whether the differences in MAP

	HARMTRACEALIGN	TRIADALIGN	CSAS	TPSD	LCESSIM	LCESSIZE	BASELINE
MAP	0.722	0.711	0.696	0.580	0.585	0.422	0.459

Table 5.1: The mean average precision of the rankings based on the compared similarity measures.

are significant by performing a non-parametric Friedman test with a significance level of $\alpha = 0.01$. We chose the Friedman test because the Friedman test does not assume a specific distribution of variance.[25] To determine which pairs of measurements differ significantly we conducted a post-hoc Tukey HSD test. This way of significance testing is standard in MIREX.

The MAP scores are presented in Table 5.1 and Figure 5.3 displays the interpolated precision and recall. The fact that all lines have a similar shape is due to the very small group sizes. Within the dataset most queries have only one, two, or three relevant songs to search for (see Table 2.7). Hence, the precision is calculated at a very limited number of recall positions, yielding a similar shape for each algorithm.

There are significant differences between the runs, $\chi^2(6, N = 1775) = 2456$, $p < 0.0001$. The pairwise significance is presented in Table 5.2; most differences between the tested similarity measures can be considered statistically significant. Only the differences between HARMTRACEALIGN and TRIADALIGN, between CSAS and TRIADALIGN, and between LCESSIM and the TPSD are not statistically significant. We can conclude that HARMTRACEALIGN is clearly the best performing similarity measure. Because the difference between HARMTRACEALIGN and TRIADALIGN is not statistically significant, we cannot conclude that adding the harmonic annotations significantly improves the alignment approach presented in Section 5.3.1. Nevertheless, it does significantly improve the overall matching results, for the HARMTRACEALIGN significantly outperforms the CSAS, and the TRIADALIGN does not. The latter is not surprising, since the CSAS and the TRIADALIGN are very related similarity measures.

All similarity measures except for the LCESSIZE outperform the BASELINE string matching approach. This clearly shows that the size of the LCES alone is not a feature distinctive enough for large scale harmonic retrieval. It also demonstrates that the *real life* dataset such as the one used in this experiments exhibits very different dynamics than the much smaller and less noisy dataset used in Section 3.4. The effect of weighting the matching between two harmonic analysis trees, as done in LCESSIM, has a strong effect which is reflected by the large difference in performance between LCESSIM and LCESSIZE. The performance of the LCESSIM is about the same as the

[25] All statistical tests were performed in Matlab 2011a.

	TRIADALIGN	CSAS	TPSD	LCESSIM	LCESSIZE	BASELINE
HARMTRACEALIGN	−	+	+	+	+	+
TRIADALIGN		−	+	+	+	+
CSAS			+	−	+	+
TPSD				+	+	+
LCESSIM					+	+
LCESSIZE						+

Table 5.2: The pairwise statistical significance between all similarity measures. A + denotes a statistically significant difference and a − denotes a non-significant difference. The + and − signs were derived by pairwise comparison of the confidence intervals.

performance of the TPSD and there is no significant difference in MAP.

5.6 Discussion

In this chapter presented four novel approaches to harmonic similarity. Two of these measures were based on the largest common embeddable subtree and two measures were based on sequence alignment. We compared the four similarity measures with the three other similarity measures introduced in Chapter 2 in a retrieval experiment using the chord sequence corpus presented in Section 2.3.1. The results demonstrated that using information about the function of a chord can improve harmonic similarity and that HARMTRACEALIGN is the best performing similarity measure tested on the Internet chord sequence corpus. The Haskell code of the newly introduced similarity measures is available online[26].

In general we can conclude that sequence alignments approaches are very suitable for defining harmonic similarity. The three best performing algorithms were all based on sequence alignment. Although adding harmonic annotations to TRIADALIGN did not yield a statistically significant improvement, HARMTRACEALIGN did outperform the CSAS significantly where the TRIADALIGN did not. Hence, we can conclude that harmonic analysis information does improve harmonic similarity estimation. However, not all harmony annotations appear to be beneficial. Although annotating chords with their functional categories (*Ton*, *Dom*, *Sub*) does not have a negative effect on the similarity estimation, it does not improve the harmonic similarity either. Perhaps the categories are not distinctive enough to be advantageous. Furthermore, it is important to realise that also both the TRIADALIGN as well as the CSAS incorporate a considerable amount of musical information. They generalise over different types of chords and

[26] package harmtrace-1.0 `http://hackage.haskell.org/package/HarmTrace-1.0`

use a key relative representation. Nevertheless, it is surprising that these relatively simple approaches yields such competitive results.

We can discern at least four factors that influence the retrieval performance. First, an important reason that the LCESSIM and the TPSD do not perform as well as the alignment based similarity measures might very well be that the LCESSIM and the TPSD do not take into account elements (chords, or nodes in a harmonic analysis tree) that are *not similar*. The non-overlapping parts of the step functions (the non-matching parts of the chord sequences) are ignored and also there is no penalty of mismatching nodes in the LCESSIM. The alignment approaches, on the other hand, have clear insertion and deletion penalties. A second factor that has a large effect on retrieval performance is the duration of the compared elements. In an alignment approach chords are compared at the beat level. Local alignment therefore allows for a very fine-grained weighting of the compared chords, because chords can be shortened or lengthened per beat with the insertion or deletion operations. Third, a factor that affects the similarity estimation was the normalisation of the similarity measure. For instance, he TPSD is normalised by the length of the shortest step function and the LCESSIM is normalised by the self similarity of both original trees. We found in all cases, that a normalisation such as applied in the four similarity measures introduces in this chapter, e.g. Equation 5.6, outperforms a simple normalisation by the length of the sequence or no normalisation at all.

Last, we noticed that similarity measures that are very specific about what is regarded similar perform best. The retrieval task of Section 5.5 can be considered a difficult one for the song class sizes are very small. Often there is only one related piece in the corpus, and finding it based on its harmony alone is challenging. We noticed that a lot of pieces exhibit very similar harmonic patterns and when one is using only harmony to search for one or two harmonically related songs among about 5000 different pieces, similarity measures that are specific perform best. With specific we mean that individual properties of chords should only be considered similar when they describe the exact same phenomenon. For example, one could consider all secondary dominants as similar, e.g. a V/V is similar to a V/II because they involve the same kind of preparation. However, in our experience retrieval performance improves if only the fifth leaps between the exact same scale degrees are considered as similar, e.g. a V/V should only be considered similar to a V/V and not to a V/II.

We believe that performing a cover-song-finding experiment as described in the previous section is a sound way of evaluating of harmonic similarity, since nothing else could have influence the results but the chords available in the data. Nevertheless, it is stimulating to think about other ways of evaluating harmonic similarity that go beyond the concept of a cover-song. A fundamental problem is that currently there

is no good ground-truth that actually captures the harmonic similarity on a gradual (non-binary) scale. But how should such a ground-truth be established: by performing a large scale user study, or by consulting musical experts? These questions remain unanswered, and pose challenges for future MIR research.

The potential of harmony models to improve harmonic similarity, as demonstrated in this chapter, supports the claim that knowledge-based models about music are important for taking content-based MIR methods to a next level. Moreover, that this does not hold for harmonic similarity alone will be demonstrated in the following chapter.

Chapter 6

Improving automatic chord transcription from audio using a model of tonal harmony

NOWADAYS, chord labels are an indispensable an ubiquitous aid for modern musicians. Although classically trained performers still mainly rely on printed scores, describing in high detail how a piece of music should be performed, the emergence of jazz, improvised, and popular music gave rise to the need for more flexible and abstract representations of musical harmony. This lead to a notational vehicle often referred to as a *lead sheet*. A lead sheet typically contains only the melody of a composition accompanied with the essential harmonic changes denoted with chord labels. It can best be considered a rather informal map that guides the performers and specifies the boundaries of the musical playground. However, also in music theory, music education, composition and harmony analysis chord labels have proved to be a convenient way of abstracting from individual notes in a score. Hence, these days chord labels are omnipresent: there are publishers that specialise in publishing lead sheets, the Band-in-a-Box software suite uses chord labels to generate accompaniment, and endlessly many lead sheets circulate on the internet.

The many possible applications of chord labels, both as an analysis, composition and educational tool as well as a convenient way to transfer and represent musical information, have sparked research focussing specifically on chord labels. For example, in the automatic harmony analysis, extensively covered in Chapter 4 (see also: De Haas et al. 2011a), and the estimation of similarity, covered in the Chapter 2 and Chapter 5 (see also: De Haas et al. 2011c,b), chord labels are used as primary data representation. However, besides that using chords as information representation is interesting from a musical perspective, it also offers clear benefits from a Music Information Retrieval (MIR) perspective. Many MIR tasks, like similarity estimation, genre detection, or query by humming, require some reduction of the raw audio signal into a manageable symbolic representation. Ideally, a piece of audio would be automatically transcribed into a representation similar to a musical score. However, although much progress has

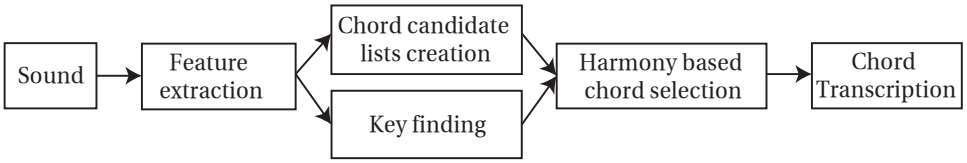


Figure 6.1: A schematic outline of the MPTREE system.

been made, multiple fundamental frequency (F0) estimation, the holy grail in polyphonic music transcription, is still considered too unreliable and imprecise for many MIR tasks. Hence, chord recognition has offered a welcome alternative, which transforms polyphonic audio into musically feasible symbolic annotations, but can used for serious MIR tasks, like the ones described in other chapters of this thesis.

In this chapter we elaborate on the automatic transcription of chord labels from polyphonic musical information. Thanks to statistical methods like Non-negative Matrix Factorization (NMF) the quality of both fundamental frequency (F0) transcription (Vincent et al. 2010) as well as chord transcription methods have seen considerable improvements. Still, we strongly believe that there is a limit to what can be extracted from the data alone and certain musical segments can only be annotated correctly when the surrounding musical context is taken into account as well. Often Hidden Markov Models (HMMs), or other generic machine learning techniques, have been assigned the task to model the transitions between chords. As we explained in Section 1.2, application of data oriented approaches to music has its limitations. Moreover, the majority of these systems only model the transition between one or two subsequent chords. The HARMTRACE harmony model (Section 4.2), on the contrary, is explicitly designed for modelling the relations between chords, also over a longer time span. Hence, in this chapter we show how the HARMTRACE harmony model can be used to improve chord recognition.

Contribution. We present MPTREE[27]: a system that introduces a novel way of improving automatic chord transcription from musical audio by employing a formal model of tonal harmony. Using the Vamp plugin architecture we extract spectral and pulse features from the raw audio signal. Next, given a dictionary of chords, we estimate the probabilities of the chord candidates matching a segment. When the spectrum clearly favours a particular chord, that chord is chosen to represent that segment. However, if multiple chords match the spectrum well, due to noise, silence, or other obscurations of the musical information, we let the harmony model decide which of the chord

[27](Musical) Model Propelled TRanscription of Euphonic Entities

candidates fits the segment best from a music theoretical point of view. We show that this approach yields a significant chord transcription improvement on the corpus of all Beatles songs.

An global outline of the system is presented in Figure 6.1. We start by reviewing the most important literature in Section 6.1. Next, we will give a complete outline of the MPTrEE system in Section 6.2 and also briefly expand on the signal processing front-end in Section 6.2.1. In Section 6.3 and Section 6.4 we discuss the experiments and results. Finally, we conclude this chapter by recapitulating the main advantages and disadvantages of the MPTrEE system and highlight some directions for future research.

6.1 Related work

Automatic chord transcription from musical data has been intensively studied from the nineteen seventies on. Although the extraction of chord labels from musical scores is interesting and has received considerable attention (e.g., Pardo and Birmingham 2002), in this chapter we focus on the extraction from chord labels from musical audio. Nevertheless, some techniques might also be applicable to the symbolic domain. We briefly highlight some of the most important contributions in this section, but for an elaborate review of the related work on automatic chord transcription we refer to (Mauch 2010).

The first computational approaches to automatic chord transcription from musical audio emerged at the end of the nineties of the previous century. The first system that extracted chord labels from an audio signal was developed by Fujishima (1999). In general, the outline of Fujishima's system is not so different from the chord transcription systems nowadays developed and also no so different from the system here presented. First, the audio signal is split into a series of overlapping *frames*. A frame is a finite observation interval specified by a certain windowing function, e.g. Hann, Hamming, Blackman-Harris, ect. Subsequently, the spectrum X_k, i.e. the amplitude of the k frequency components present in the audio signal, of the input sample x is computed for each frame with a discrete Fourier transform (DFT)

$$X_k = \sum_{n=0}^{N-1} x_n e^{-j\frac{2\pi}{N}kn}, \qquad 0 \leqslant k \leqslant N-1$$

where N is the number of samples in a frame and j is the *imaginary unit*.[28] Next, it is common to create pitch class profiles (PCP), which are also called *chroma vectors* (Wakefield 1999). A chroma vector represents the intensities of the twelve different pitch classes and can be calculated for each pitch class by summing all amplitudes of the frequency bins related to an instance of that pitch class. Finally, the chroma vectors are matched with binary chord profiles, which in Fujishima's case this is done with an euclidean distance. The chord structure that best matches the chroma vector is selected to represent the frame. Although the digital signal processing specific parameters may vary, most approaches towards automatic chord transcription use a chroma vector based representation and differ in other aspects like, chroma tuning, noise reduction, chord transition smoothing and harmonics removal. A nice overview of the different types of chroma features used in chord labelling is presented in (Jiang et al. 2011) and for an interesting discussion on the interdependency of the various components of a chord labeling systems we refer to (Cho et al. 2010).

From 2008 on, chord transcription has also received a considerable amount of attention in the yearly benchmarking challenge MIREX (Downie 2008; Downie et al. 2010). Each year, between 7 and 15 different chord transcription algorithms were evaluated on a dataset. In 2008, the system of Bello and Pickens (2005), which was the first to synchronise chroma vectors at every beat, did very well. The following year, Mauch et al. (2009) presented a system that gave good results by structurally segmenting a piece and combining chroma information form multiple occurrences of the same segment type. Again, in 2010, Mauch and Dixon (2010) improved their previous results by using an approximate note transcription technique that is explained in Section 6.2.1.

6.2 System outline

In this section we will give a general outline of the MPTrEE system by explaining the different elements of the detailed flowchart shown in Figure 6.2. First, we extract chroma features and beat locations from the audio signal and synchronise the chroma features at the beat positions. The chroma features are used to estimate the global key and possible modulations in the musical audio and for creating a sequence of chord candidate lists. If there is a lot of uncertainty in the data, this list might contain multiple chords; however, when there is a strong match between the spectrum and one particular chord candidate, this chord will be the only candidate. Subsequently, the sequence of chord candidate lists is segmented by key, and by the chords on I and V relative to the local key. Finally, the best matching sequence per segment is selected

[28] explaining complex numbers and the Euler formula is beyond the scope of this chapter, see for instance (Smith 1997).

by expanding all possible sequences, and preferring the simplest sequence with least errors.

6.2.1 Feature extraction front-end

The research presented in this chapter heavily depends on the so called Vamp plugin architecture[29] as a feature extraction front-end. Akin to Virtual Studio Technology (VST) effect plugins, Vamp plugins can be loaded by a host application to transform an audio signal. However, unlike an effect plugin, a Vamp plugin does not return audible information, but instead returns numerical information that provides visualisable insights into the structure of the audio signal. Typically, a Vamp plugin might calculate points in time, such as possible beat positions or note onsets; various forms of audio spectrograms; or curve data, like the most prominent frequency per time frame.

As feature extraction front-end we rely on the NNLS Chroma Vamp plugin[30] developed by Mauch (2010). The NNLS Chroma plugin transforms an audio signal into two 12-dimensional chroma vectors representing the harmonic content at each frame. The first *bass* chroma vector represents the low notes and emphasises on the lower frequencies and the second *treble* chroma vector represent the higher notes emphasising on higher frequencies. The idea behind this separation is to model the prominent role of the bass note in chord transcription. We present a brief overview of the most important properties and parameters of the NNLS plugin, but for specific signal processing details we refer to (Mauch 2010, Chapter 3 and 5).

We use the sonic-annotator[31] (version 0.5) as Vamp host and sample the audio audio tracks at 44,100 Hz. If the audio file contains two stereo channels, the mean of both channels is used for analysis. We set the sonic-annotator to use a Hann window of 16,384 samples and a hop size, i.e. the amount of samples that overlap between two subsequent frames, of 2,048 samples. Next, the spectrogram is calculated at each frame using a discrete-time Fourier transform and mapped to a spectrogram with bins that are linearly spaced in log-frequency (very similar to a constant-Q transform; Brown 1991). A log-frequency scale fits musical audio better than using bins that are separated by a constant frequency because pitch behaves linear in log-frequency.

The NNLS Chroma Vamp plugin also accounts for tuning differences in the audio signal. In the human perception of musical pitch the relative differences between notes play a more important role than the absolute frequencies of the notes. Many instru-

[29] `http://www.vamp-plugins.org`
[30] `http://isophonics.net/nnls-chroma`
[31] `http://omras2.org/SonicAnnotator`

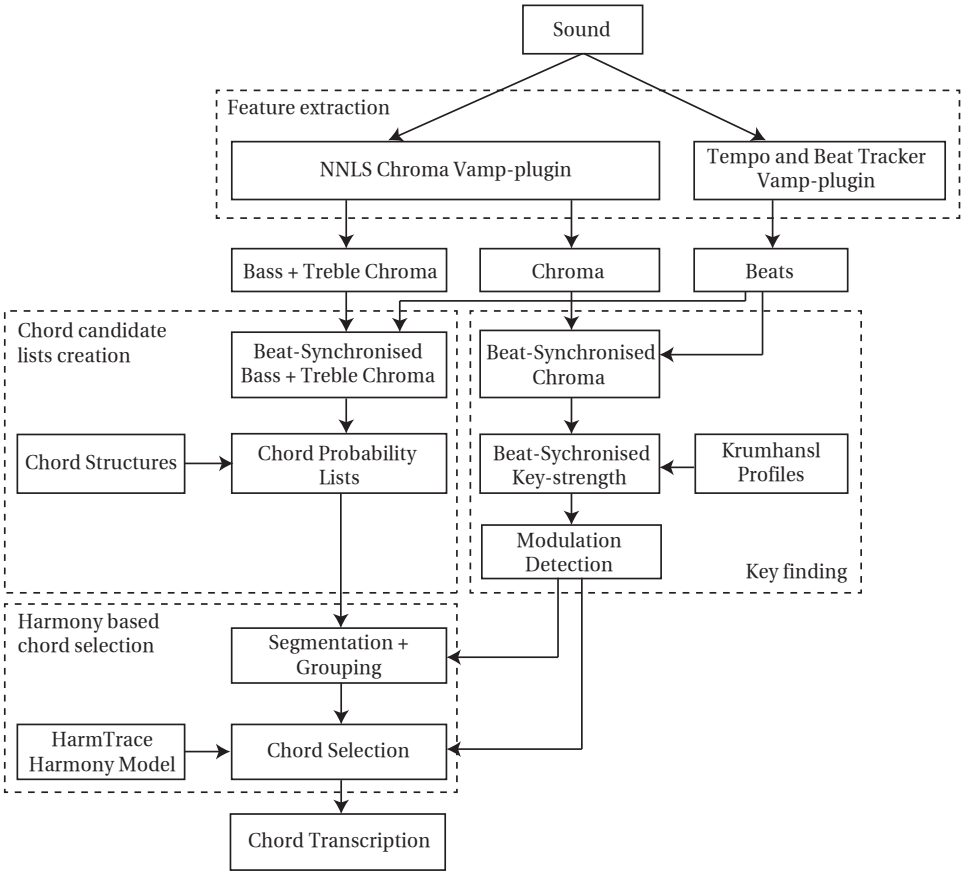


Figure 6.2: A schematic outline of the MPTREE system. The boxes with dotted lines denote the high level modules as outlined in Figure 6.1.

ments can be tuned to match other instruments in an ensemble such that within the ensemble the absolute frequencies can be synchronised. Nevertheless, different reference frequencies are used; typically, for the A above middle C a reference frequency of 440 Hz is used. To account for tuning differences in recordings (assuming equal temperament), two gromagrams with 36 bins, three bins per semitone, are calculated. Based on the relative magnitude of the three bin classes, the log-frequency spectrogram is updated by linear interpolation such that the centre bin is in concordance with a 440 Hz reference tuning. Finally, the three bins representing one semitone are averaged.

Generally, a musical tone consists of a fundamental frequency and its *overtones*, which can be *harmonics*, i.e. partials with frequency values that are integer multiples of the fundamental frequency, or other non-harmonic partials. Not every harmonic has to be present in the signal (including the first and fundamental harmonic): this depends on the type of sound or instrument. The goal of the NNLS chroma plugin is to estimate which pitch activation best matches the interference of the partials found in a spectrum. To account for variety of activation patterns, a semitone-spaced note activation matrix, N, is created in which every row represents a note and every column a frequency bin. In N notes are modelled with exponentially decreasing partials. Next, every log-frequency spectrogram S can be viewed as a linear combination, $S \approx Nx$, where x is the pitch activation vector (which is, in a sense, very comparable to NMF). Subsequently, x can be found by minimising the euclidean distance: $\| S - Nx \|$, where $x \geqslant 0$. This problem is known as the non-negative least squares problem, which can solved by an algorithm proposed by Lawson and Hanson (1974).

6.2.2 Beat-synchronisation

After obtaining a bass and treble chroma feature we synchronise both chroma features by averaging the features vectors between two beats. For this we obtain a list of beat positions by utilising another Vamp plugin: the Queen Mary, University of London, Tempo and Beat Tracker plugin (Davies and Plumbley 2007).[32] This beat tracker uses a complex domain onset detection function (Duxbury et al. 2003), which combines difference-in-energy-based onset detection with a phase-based onset detection. Once the estimates of periodicity have been determined, the actual beat positions are recovered by applying Ellis' (2007) dynamic programming algorithm. Having the actual beat locations, the matrix of chroma vectors is segmented at every beat position and a single chroma feature per beat is obtained by averaging the feature vectors with each segment.

[32] `http://vamp-plugins.org/plugin-doc/qm-vamp-plugins.html#qm-tempotracker`

									0.93	C^7										
									0.96	Am										
						0.94	G		0.97	C						0.94	B^0			
...	1.00	C	1.00	F	1.00	Gm		1.00	Em		1.00	F		1.00	B	...				

Table 6.1: An excerpt of a sequence of chord candidate lists. The number left of the chord label represents its normalised euclidean distance with the current beat aligned chroma vector.

6.2.3 Chord probability estimation

To be able to use the HARMTRACE harmony model, we require the key, a list of chords, and a probability that the current frame actually represents that chord for every beat segment. To estimate the probability of a particular chord sounding at a beat position, we assemble a dictionary of chords that we expect to be able to occur in the music. A chord is represented as a binary 12-dimensional vector where the first position denotes the root of a chord and the remainder of the positions denote the other simultaneously sounding pitches. This allows us to model any possible chord within one octave. Currently, we use a limited chord dictionary with four chord structures: major, minor, diminished seventh and dominant seventh. We chose these four chords, because they map exactly to the four chord classes used by the HARMTRACE harmony model (see Chapter 4, page 65). The bass note of the chord is modelled with an additional 12-dimensional vector containing only one pitch for the bass note to match the bass chroma vector as outputted by the NNLS chroma plugin. Next, the we generate the chord dictionary by cyclically rotating all chord structures for all twelve semitones, yielding 48 different chord candidates.

Having a matrix of beat-synchronised bass and treble chromagrams and a chord dictionary, the probability of a chord sounding at a particular beat is estimated by calculating the euclidean distance between the chord structures and the chroma feature. These distances are calculated for every chord candidate at every beat. Next, we sort the chords candidates by descending euclidean distance. To obtain a relative measure of the fit between a chord candidate and the chroma vector in the range $[0,1]$, the distances are normalised by dividing them by distance of the best matching chord candidate. In case the information in the spectrum clearly favours a certain chord candidate, the initial differences in normalised distances will be relatively large and will decrease quickly after the head position. Hence, we can use these differences as a measure of relative chord candidate preference. If this preference is very strong, the top chord candidate will be selected to represent that beat, if this preference less pronounced we let the HARMTRACE harmony model decide which of the chord candidates would harmonically make most sense. Typically, this results in a sequence of chord candidates

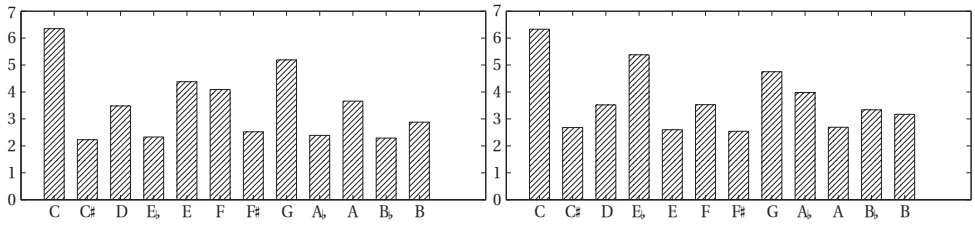


Figure 6.3: The Krumhansl-Kessler profiles (1982), for C major (left) and C minor (right).

similar to the one shown in Table 6.1. The selection is performed by cutting the chord candidate list off at a fixed distance. The cut-off value is an important parameter to the model, both influencing the speed and the transcription performance of the system. After a some experimentation we found a cut-off value of .92, to give good results. The chord candidate sequences will form the bases for our high-level selection mechanism.

6.2.4 Key-finding

To be able to use the HARMTRACE harmony model for the selection of chord sequences that are music theoretically realistic, we require information about the key of the piece. To fulfil this requirement, we present a key-finding algorithm inspired by the ideas of Temperley (2001, Chapter 7) and Krumhansl (2001, Chapter 4). Again, for feature extraction we depend on the NNLS chroma Vamp plugin, which allows for exporting different kind of audio features. For the key finding we export a single tuned chroma feature without the NNLS pitch activation estimation. The tuning and time-frequency transformations are performed as described in Section 6.2.1.

To estimate the key of a piece and the possible modulations, a key-finding a *key-profiles* based algorithm is presented. A key-profile is a 12 value vector representing the stability of the twelve pitch classes relative to a given key. The values of these profiles (see Figure 6.3) are based on empirical measurements of Krumhansl and Kessler (1982, see also Section 2.1.3), in which subjects were asked to rate the how well a pitch class fits a previously established tonal context on a 1 to 7 scale. Given the major and minor key profiles, a key-strength table, K, is created; this table stores at every beat position the estimated strength of all 24 keys. The key strength is estimated by calculating the Pearson correlation coefficient, r, between a chroma vector and all 24 key-profiles:

$$r = \frac{\sum(x-\bar{x})(y-\bar{y})}{\sqrt{\sum(x-\bar{x})^2 \sum(y-\bar{y})^2}}$$

Here, x denotes the input chroma vector and y a key-profile; $\bar{x}$ and $\bar{y}$ denote the average values of the chroma vector and the key-profile, respectively. The value of r close to 0 indicates that there is little to no relation between the key-profile and chroma and a value close to 1 or -1 indicate a positive or negative linear dependence between the key-profile and the chroma vector, respectively.

Matching the key-profiles at every beat does not yet result in the desired key assignment; because beat size segments are rather small, key changes occur too often. To overcome this problem, a simple dynamic programming algorithm based on the algorithm in (Temperley 2001, Chapter 7) is used to smooth the key changes. We create a table M storing the cumulative key-strength of every key at every beat and minimise the number of modulations. This behaviour is captured in the following recurrent formula:

$$\begin{aligned} M[0,j] &= K[0,j] \\ M[i,j] &= \max \begin{cases} M[i-1,j]+K[i,j], \\ M[i-1,j]+K[i,k]+p, \qquad \{k \mid \forall x : K[i,x] \leqslant K[i,k]\} \end{cases} \end{aligned}$$

Here, M stores the cumulative key-strength for every i^{th} beat and every j^{th} key. Similarly, K stores the correlation between every i^{th} chroma vector and j^{th} key, as defined above and k denotes the index of the best matching key at beat i. The parameter p specifies the modulation penalty. We found a value of 1 for p to give good results. Finally, we obtain the definite key assignment by keeping track of the maximum cumulative key-strength at every beat.

6.2.5 Segmentation and grouping

The general idea is, given a sequence of chord candidate lists, to parse all possible combinations of sequences of chords with HARMTRACE (see Chapter 4) and select the simplest analyses with the least amount of errors. However, the number of possible combinations grows exponentially by the number of candidate lists with a length greater than 1. Furthermore, this would imply parsing rather long and very similar sequences. As a consequence, the parser must repeatedly parse the same subsequences, making the process as a whole even more computationally expensive. Hence, it is vital to split our sequence of chord candidate lists into smaller musically meaningful segments.

Also, from a musical point of view it is unrealistic to expect chords to change at every beat. Therefore, we obtain a first reduction of the space of analysed sequences, by

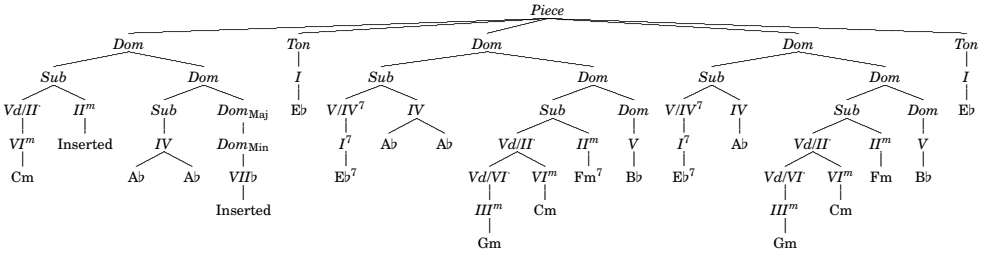


Figure 6.4: An excerpt of the HARMTRACE analysis of *The long and winding road* by the Beatles (of which the ground-truth chord annotations were used for parsing).

merging chord candidate lists that contain the same chords, regardless of their order. After all, the chords were collected in a candidate list in the first place because the chroma vector was not pronounced enough to yield a clear order. Next, the sequence of chord candidates is segmented on the basis of the estimated key, resulting in segments that contain only a single key assignment. Nevertheless, these sequences are still rather long to be parsed directly.

In Figure 6.4 an excerpt of the harmony analysis of *The long and winding road* by the Beatles is depicted. Within the HARMTRACE harmony model, a piece is viewed as a collection of tonics (*Ton*) and dominants (*Dom*) nodes. Hence, from the parsing point of view, splitting a chord sequence into segments that match the subtrees rooted by a *Ton* or *Dom* seems natural. Because local information about the key is available, we can calculate the key relative scale degrees of the chords and split a sequence at every beat where a I or V occurs in a chord candidate list. This gives us sequences that are short, but still musically meaningful. In case our key finding method is off the mark and we still end up with a rather long sequence, we enforce the sequences to be no longer than 8 chords and and expand into no more than 20 different candidate sequences.

6.2.6 Chord selection by parsing

Now that we have access to both a segmented sequence of chord candidate lists and local key information, we are ready to apply the HARMTRACE harmony model. For every segment we parse all possible combinations of chord sequences and select the sequence that has the lowest error-ratio. The error-ratio is the number of insertions and deletions of the error-correcting parser (see Section 4.2.2) divided by the number of chords. When two sequences have the same error-ratio, we select the most simple solution by picking the sequence that returns the smallest parse tree. In case also the parse tree size is identical, we select the sequence returning the parse tree of least

depth.

The harmony model used in MPTrEE is exact same model as the one described in Chapter 4 and no adjustments were made. If we would change the model, we should also thoroughly document and motivate these changes and measure their effects. Hence, for sake of simplicity we choose not to alter the model. Nonetheless, there are sound reasons for changing the model. For instance, the HARMTRACE harmony model exhibits a bias towards jazz harmony, perhaps adding some specifications for explaining certain blues phenomena, might suit the Beatles corpus. Furthermore, the model, as specifically described in Section 4.2, is tuned to be very robust against very long and possibly highly ambiguous chord sequences. Since the a sequence of chord candidates have a maximum length of 8 chords, there is no lurking combinatorial explosion of ambiguous solutions. Hence, we could have relaxed some model specifications and constrains making the model more general, but allowing multiple specifications to explaining the same harmonic phenomena.

6.3 Experiments

To measure the effect of the various modules on the chord transcription performance, we evaluate four different versions of the system described above. The first and most simple system we name SIMPLE and always selects the chord that best matches the bass and treble chroma vectors. The second system, GROUP, also only considers the best matching chord candidate, but does incorporate the grouping as described in Section 6.2.5. The third system is the full MPTrEE system, including key finding as described in Section 6.2.4, as described above. Finally, a forth system is evaluated to measure the effect of the key finding. This system, $\text{MPTrEE}^{\text{key}}$, does not use the key finding, but instead uses ground-truth key annotations. All four systems are implemented in the functional programming language Haskell and compiled using the Glasgow Haskell compiler version 7.02.

We evaluate the quality of an automatic chord transcription by comparing it to a transcription of the same piece made by a human expert. Unfortunately, not many reference chord annotations are available. Hence, because of the availability of high-quality ground-truth annotations (Mauch et al. 2009),[33] we evaluate our system on the 179 songs from 12 Beatles albums. The reference annotations contain detailed chord labels stored in the non-ambiguous chord notation proposed by (Harte et al. 2005). Generally, this syntax allows for expressing hundreds of different chords. However, automatic chord transcriptions systems, such as MPTrEE, have a much more limited chord vocab-

[33] `http://isophonics.net/content/reference-annotations`

	SIMPLE	GROUP	MPTREE	MPTREE^{key}
RCO	0.653	0.687	0.711	0.715
Running time	7 : 42	8 : 00	62 : 25	71 : 00

Table 6.2: The relative correct overlap and the running times (minutes : seconds) for the baseline, MPTREE and MPTREE ^{key} chord transcription systems.

ulary. Therefore, in the MIREX evaluation the chord vocabulary is limited to 24 major and minor chords augmented with a "no chord" chord, enabling us to classify also non-musical sound. The translation from the four chord classes of HARMTRACE to major and minor chords is trivial: chords of the major and dominant class are classified as major, and chords of the minor and diminished seveth class are classified as minor.

Typically in MIREX, the *relative correct overlap* (RCO) is used as a measure of transcription accuracy. The RCO is defined as:

$$RCO = \frac{\text{total duration of correctly overlapping chords}}{\text{total duration}}$$

Both the ground-truth and the automatic chord transcription consist of a chord label in the syntax defined by (Harte et al. 2005) and an accompanying onset and offset timestamp. We approximate the RCO by sampling both the ground-truth and the automatic annotations every 10ms and dividing the number of correctly annotated samples by the total number of samples.

6.4 Results

We have compared the MPTREE, MPTREE^{key}, GROUP, and the baseline SIMPLE system on 179 songs of the Beatles. All runs were performed on the same Intel Core 2 6600 machine running at 2.4 GHz and equipped with 3 GB of random access memory. The measured differences in RCO and running times are displayed in Table 6.2.

We tested whether the differences in RCO are statistically significant by performing a non-parametric Friedman test[34] with a significance level of $\alpha = 0.05$. The Friedman ANOVA is chosen because the underlying distribution of the RCO data is unknown, and, in contrast to a regular ANOVA, the Friedman does not assume a specific distribution of variance. To determine which pairs of measurements differ significantly a

[34] All statistical tests were performed in Matlab 2011a.

post-hoc Tukey HSD test is conducted. Within the MIREX challenge the same statistical procedure is followed. There are significant differences between the different systems, $\chi^2(3, N = 179) = 293$, $p < 0.0001$. Not all pairwise differences between systems are statistically significant; the difference between MPTrEE and MPTrEE$^{\text{key}}$ was not significant, all other pairwise difference can be considered statistically significant.

Considering the differences between the MPTrEE system, with or without key finding, and the SIMPLE and GROUP systems, we can conclude that using the HarmTrace harmony model for chord candidate selection improves chord transcription performance. This difference in performance cannot be attributed to the grouping function described in Section 6.2.5 alone; although grouping significantly improves transcription performance, also the difference between GROUP and MPTrEE is statistically significant, which can only be due to the model based chord selection.

In interesting observation is that there is no statistically significant difference between the MPTrEE system using the ground-truth key annotations and the key finding. We expected the difference between the MPTrEE and MPTrEE$^{\text{key}}$ systems to be larger because there is always a margin of error in key finding. Nevertheless, the keys found by key finding algorithm described in Section 6.2.4 does not find keys that are harmful for the system as a whole. This does not necessary mean that it is always finding the correct key. To measure the performance of the key finding on its own the automatic key transcriptions should be systematically compared with the ground-truth annotations in a similar way as has been done with the chord transcriptions.

The running times as shown in Table 6.2 exclude the time taken by the Vamp feature extraction plugins. The results show a clear trade off between transcription performance and computation time. However, the running times are acceptable, about 45 seconds per song on average. The difference in runtime between the MPTrEE and MPTrEE$^{\text{key}}$ is surprising; we cannot directly explain this difference, but it is clear that the time taken by the key finding process has little to no effect on the total running time.

6.5 Discussion

In this chapter we aimed at bridging the gap between bottom up audio feature extraction and top-down symbolic music analysis by demonstrating in a proof-of-concept that automatic chord transcription can be improved by using a model of tonal harmony. The feature extraction we put out to the NNLS chroma and the Tempo and Beat Tracker Vamp plugin and perhaps that using different signal processing parameters, or using different plugins, might have improved the results, but showing that was not the aim

of this chapter. We demonstrated that selecting combinations of chords that were music theoretically realistic yields better chord transcription than just picking the best matching chord at each beat, even after smoothing the chord changes with a grouping function. This result can be considered good, because, as typically done in a proof-of-concept, we only connected the different technologies without tuning their parameters. Hence, we expect that carefully tuning the parameters and tailoring the modules to maximise their interoperability will result in an increase of performance. The MPTREE Haskell code can is part of the HARMTRACE system and can be downloaded online[35].

The current state-of-the-art in automatic chord transcription is the system of Mauch (2010), who devoted his PhD thesis to this topic. He evaluated various variants of his system on the dataset of the MIREX 2009 challenge, which consists of 174 Beatles songs, 18 Queen songs, and 18 Zweieck songs. The difference in data[36] makes it difficult to compare the results of this chapter with his state-of-the-art system in an absolute manner. Ideally, this should be done in an next iteration of the MIREX challenge. Matthias Mauch's methods score an RCO between 0.655 and the 0.807 depending on the configuration and used methods. His best configuration incorporates non-negative least squares estimation (similar to the Vamp plugin described in Section 6.2.1), a dynamic bayesian network to model the chord transitions, and a structural segmentation algorithm. Since the bayesian network contains a large number of configurable parameters, it is unclear how much overfitting could have taken place. Clearly, no overfitting has taken place in the systems here evaluated, since both the model and the signal processing front-end have not been tuned to improve the results. We are confident that further tuning of the parameters of our system will yield a comparable performance that we can demonstrate in a future edition of the MIREX challenge.

A clear benefit of the model-based approach presented in this chapter, opposed to the often used HMMs, is that we can analyse why certain chords were preferred over others and reason about whether this choice was justified. While, an HMM, as many other generic machine learning approaches, remains a black box, that does not tell the researcher or end-user anything about the choices made. We do think that it is important to evaluate a more thoroughly tested version of MPTREE on larger corpora and compare these results with other approaches, including systems based on machine learning techniques.

Nonetheless, various aspects of the system can be improved. In the current MPTREE

[35] package harmtrace-1.0 `http://hackage.haskell.org/package/HarmTrace-1.0`

[36] besides the different songs, also different remastered editions of the Beatles albums exists. Of some of these recordings it is known that they deviate from the ground-truth files as provided by the Queen Mary university of London. It is unclear if this has influenced the results presented in this chapter. However, if this is the case, the results of all compared systems are affected equally.

system we used the HARMTRACE harmony model, which exhibits a bias towards jazz harmony. Adapting the harmony model specifically to the style of the corpus used will very probably improve the results. Additionally, Mauch et al. (2009) successfully incorporated a structural segmentation into its chord transcription system. He improved the chord recognition by averaging the chroma vectors of segments that were classified as having very similar harmonies. This technique could possibly improve the results in the MPTREE system as well. Although we consider the running times to be acceptable, the running times may be decreased by setting up a chord sequence database. Such a database should store the parse results for frequently occurring chords sequences, which then only have to be parsed once.

In general, we demonstrated that connecting state-of-the-art low-level feature extraction methods to high-level symbolic knowledge systems offers a challenging combination which is applicable to many common retrieval tasks, such as cover-song finding, music transcription, structural analysis, and offers new capabilities to boosts the research of analysis and retrieval of musical audio.

Chapter 7

Conclusions and future research

THROUGHOUT this doctoral dissertation we have illustrated the advantageous role of chords and chord sequences in organising, analysing and representing harmonic music information. We showed that chord sequences present fruitful musical representations that abstract from individual notes in a score and that can be extracted from musical audio with considerable accuracy. As we elaborately demonstrated, chord sequences are very suitable notational vehicle for determining harmonic similarity, and analysing the structure of a harmony progression in a piece of music. The exploitation of the beneficial role of chords for solving MIR tasks has been the central focus of this dissertation. In this chapter we briefly recapitulate the main conclusions and elaborate on some directions for future research.

7.1 Conclusions

In Chapter 1 we argued that there are limits to what can be learned solely from musical data and that for crushing the proverbial glass ceiling more musical knowledge should be incorporated into MIR systems. This most certainly holds for the main topics of this dissertation: the investigation of the similarity, analysis, and extraction of harmonic music information that unfolds over time.

In Chapter 2 we introduced a first geometrical approach towards harmonic similarity: the Tonal Pitch Step Distance (TPSD). We used Lerdahl's Tonal Pitch Space model (2001) to represent a chord sequence as a step function that captured the distance to the tonal centre at every beat. Although we found other measures of similarity presented in the chapters that followed to outperform the TPSD, it is fast and the results show that the TPSD could effectively be used to retrieve cover-song versions of chord sequences. For this cover-song-finding experiment a novel dataset was created that consisted of 5028 user generated chord sequences. In this experiment we also varied the amount of chord information used and a surprising finding was that using the tri-

adic chord notes and ignoring all chord additions yielded the best results. We furthermore demonstrated how a harmonic similarity measure can provide additional insights into the relations between harmony and melody in 357 chorales by J.S. Bach. However, some limitations of the TPSD encouraged us to pursue various other approaches to harmonic similarity in the chapters that follow.

In Chapter 3 we reasoned that we should compare chord sequences in a way similar to how musicians compare them and incorporate the harmonic function of a chord in the similarity assessment. In order to obtain these harmonic functions, a harmonic analysis of the chord progression at hand was needed. Consequently, we presented a proof-of-concept in which we used a generative grammar of tonal harmony based on ideas of Rohrmeier (2007) to perform an automatic harmonic analysis. Because we assumed that not every chord was equally important the harmonic functions were organised hierarchically, and an analysis had the shape of a tree. Next, we presented an algorithm for matching hierarchical harmonic annotations and used these matchings to estimate the similarity of the two compared chord sequences. The results showed that these measures can be used to retrieve harmonically related chord sequences in a small dataset of 72 jazz chord sequences. However, this grammar-based harmonic similarity also revealed some problematic issues concerning the parsing of noisy chord sequences and the matching of trees with duplicate labels. We addressed both the parsing as well as the matching issues in the two subsequent chapters.

In Chapter 4 we showed how the weaknesses of the generative grammar based approach could be overcome by applying advanced functional programming techniques, such as type-level representations and state-of-the art error-correcting parsers. The result of this endeavour was named HARMTRACE: a system that allowed us to obtain an automatic harmonic analysis and was robust against noisy data, fast, and easy to maintain. The analyses created by HARMTRACE were not only in accordance with harmony theory, but also proved them selves useful in later chapters. Both the quality and the speed were demonstrated on the chord sequence corpus introduced in Chapter 2. We furthermore discussed various example analyses and highlighted the most important Haskell implementation details.

Chapter 5 revisited the harmonic similarity matching ideas put forward in Chapter 3. In this chapter we proposed four new harmonic similarity measures; two of these similarity measures we based on the largest common embeddable subtree of two harmonic analyses, and the other two were based on local alignment. In three of these similarity measures the harmonic analyses generated by HARMTRACE played an indisputable role. We evaluated both the four newly presented similarity measures as well as the three similarity measures introduced in Chapter 2 in a retrieval experiment on the corpus presented in Section 2.3.1. The results showed that the local alignment approach

that exploited the function of a chord within its tonal context performed best, yielding a mean average precision of 0.722. This is currently the best retrieval result obtained for this corpus.

The similarity measures presented in Chapter 2 and Chapter 5 assumed the availability of sequences of symbolic chord labels. In Chapter 6 we presented a novel method for extracting these labels from digital musical audio data, today the most common representation of music. We used the Vamp plugin architecture to extract spectral and pulse features from the audio signal and used these features to obtain beat synchronised chord candidate lists. Subsequently, if the data was in strong favour of a particular chord, this chord was chosen to represent that particular segment. However, when there was uncertainty in the audio information, and multiple chords seemed to match the segment well, we let the HARMTRACE model decide which of the chord candidates was music theoretically the most realistic candidate. We evaluated four different transcription systems to measure the effect of key information, grouping, and the HARMTRACE harmony model. The results showed that the harmony model-based approach significantly outperformed the other systems on 179 Beatles songs and yielded a relative correct overlap of 0.715.

7.2 Implications and future research

The general aim of this doctoral dissertation was to deliver new insights into harmonic similarity that allowed for the development and improvement of scalable and high-quality MIR systems. This aim has been achieved, for this dissertation has presented a new harmonic similarity measure that outperforms all other harmonic similarity measures that compare sequences of chord labels. In essence, the methods presented in this dissertation, from the analysis of an audio file up to harmonic similarity estimation, are all that is needed for a harmony-based retrieval system. However, what we have not yet investigated is a retrieval system that actually connects all components presented in this dissertation. Hence, it would be interesting to link the chord transcription, harmonic analysis and similarity estimation and create, for instance, a cover-song finding system that can be evaluated in the cover-song finding MIREX task.

Although we showed that the cover-versions of chord sequences can be retrieved accurately with the methods described in this thesis, a user of this retrieval system might also be interested in songs that share musical properties other than harmony. Hence, a straight forward, but interesting extension to our harmony-based retrieval system would be the inclusion of other musical similarity measures. First and foremost, melody should be added, but also timbre or rhythmical similarity could be music-

ally satisfying additions. This directly raises questions about how one should combine these different similarity measures. How should they be weighted, and how can user feedback be taken into account? Also, it might not always be similarity that a user is looking for; perhaps a user wants to retrieve songs that share the same melody, but are harmonically very different. This requires notions of harmonic dissimilarity, which might not simply be the inverse of the similarity measures presented in this dissertation. Maybe a user is searching for surprising and not necessarily similar music. These issues present some challenging directions for future MIR research, illustrating that content-based retrieval of music is not a solved problem yet.

All these issues involved in the development of content-based retrieval systems, from feature extraction up to similarity estimation, share one common difficulty: the lack of ground-truth data. To be able to evaluate similarity measures, transcription methods, and analysis tools, we need to compare an algorithmical result to an ideal solution produced by humans manually performing the same task. This is a very time-consuming enterprise, which is often frustrated with copy-right issues, and the final result is sometimes hard to generalise to average use cases. An promising new way of gathering these desirable ground-truths may be offered by crowd-sourcing solutions, like the ones proposed in (De Haas and Wiering 2010; Honing 2010), which tempt seemingly endless amounts of anonymous Internet users to annotate musical information.

The individual methods that were covered in this dissertation also imply some interesting directions for future research. For example, it would be interesting to investigate how the largest common embeddable subtree and local alignment based harmonic similarity measures relate to the similar melodies of Bach's chorales, similar to the experiments presented in Section 2.4.1, and how these results would extend to other chorale settings, e.g. organ or cantata settings. This may provide some new musical insights that go beyond a cover-song finding experiment. All the more because it is quite likely that the different kinds of harmonic similarity proposed and evaluated in this dissertation quantify harmonic similarity in a different way that was not captured in the experiment in Section 5.5, due to the nature of the dataset and the cover-song finding task.

Similarly, it would be interesting to study the automatic harmonic analyses of other than jazz pieces. Again, Bach's chorales would be an obvious choice, but also popular music might be a challenging source because of its prominent place in nowadays societies. We expect that the core rules as specified in Section 4.2 will largely remain appropriate, but it might require some style specific extensions. Nevertheless, the model in general can be extended in several ways: modulation has not yet been fully incorporated, the grouping of tonic and dominant annotations into phrases should be arranged for, and it would be intriguing to investigate the effects of such changes on

harmonic similarity and analysis.

Although the results in Section 5.5 show that our HARMTRACE based alignment approach is the best performing harmonic similarity measure for chord sequences currently available, we still believe there might be room for improvement. As we stated in Chapter 5, the LCES based similarity measures do not penalise the parts of the chord sequences that are not similar, and it would be challenging to investigate if calculating a tree-edit-distance (Bille 2005; Magalhães and De Haas 2011) between two harmonic analysis trees, which is algorithmically very related to the LCES, would capture harmonic similarity well. Naturally, the relations between the nodes should be weighted accordingly (as explained in Section 5.3 and Section 5.4), but an advantage of a tree-edit-distance is that it quantifies both the similarity between the nodes as well as the nodes that are not matched at all.

Finally, it will be exciting to investigate the full potential of the chord transcription system proposed in Chapter 6. As Figure 6.2 shows, there are many different modules that both individually as well as in conjunction, affect a final chord transcription. It seems likely that careful tuning the parameters and tailoring the modules to boost their interoperability will significantly increase the transcription performance. For instance, we expect that augmenting the HARMTRACE harmony model with some popular music specific specifications, will improve the recognition in popular music corpora, which are currently predominant in the MIREX chord transcription task.

All in all, this doctoral dissertation has presented a major step forward in harmony based music information retrieval. Although I must admit that machines, which can listen, understand, and appreciate polyphonic music, are likely to remain out of our reach for quite a while, the recent automatic analysis and retrieval tools covered in this dissertation present a humble step that moves the field of MIR into the right direction. Hence, I expect that in the near future harmony based feature extraction and similarity estimation will be incorporated into concrete systems for the retrieval of polyphonic music.

Bibliography

Aloupis, G., Fevens, T., Langerman, S., Matsui, T., Mesa, A., Nuñez, Y., Rappaport, D., and Toussaint, G. (2004). algorithms for computing geometric measures of melodic similarity. *Computer Music Journal*, 30(3):67–76.

Arkin, E., Chew, L., Huttenlocher, D., Kedem, K., and Mitchell, J. (1991). An efficiently computable metric for comparing polygonal shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(3):209–216.

Aucouturier, J. and Pachet, F. (2004). Improving timbre similarity: How high is the sky. *Journal of Negative Results in Speech and Audio Sciences*, 1(1):1–13.

Babbitt, M. (1965). The use of computers in musicological research. *Perspectives of new music*, 3(2):74–83.

Baroni, M., Maguire, S., and Drabkin, W. (1983). The concept of musical grammar. *Music Analysis*, 2(2):175–208.

Bello, J. and Pickens, J. (2005). A robust mid-level representation for harmonic content in music signals. In *Proceedings of the 6th International Symposium on Music Information Retrieval (ISMIR)*, pages 304–311.

Bigand, E. (2003). More about the musical expertise of musically untrained listeners. *Annals of the New York Academy of Sciences*, 999:304–312.

Bigand, E. and Parncutt, R. (1999). Perceiving musical tension in long chord sequences. *Psychological Research*, 62(4):237–254.

Bigand, E. and Poulin-Charronnat, B. (2006). Are we experienced "listeners"? a review of the musical capacities that do not depend on formal musical training. *Cognition*, 100(1):100–130.

Bille, P. (2005). A survey on tree edit distance and related problems. *Theoretical Computer Science*, 337(1–3):217–239.

Bishop, C. (1995). *Neural Networks for Pattern Recognition*. Oxford University Press, USA.

Brown, J. (1991). Calculation of a constant Q spectral transform. *Journal of the Acoustical Society of America*, 89(1):425–434.

Cannam, C., Landone, C., and Sandler, M. (2010). Sonic visualiser: An open source application for viewing, analysing, and annotating music audio files. In *Proceedings of the ACM Multimedia 2010 International Conference*, pages 1467–1468, Firenze, Italy.

Casey, M., Veltkamp, R., Goto, M., Leman, M., Rhodes, C., and Slaney, M. (2008). Content-based music information retrieval: Current directions and future challenges. *Proceedings of the IEEE*, 96(4):668–696.

Celma, O. and Serra, X. (2006). FOAFing the music: Bridging the semantic gap in music recommendation. *Web Semantics: Science, Services and Agents on the World Wide Web*, 6:250–256.

Chakravarty, M. M. T., Ditu, G. C., and Leshchinskiy, R. (2009). Instant generics: Fast and easy. Draft version.

Chemillier, M. (2004). Toward a formal study of jazz chord sequences generated by Steedman's grammar. *Soft Computing–A Fusion of Foundations, Methodologies and Applications*, 8(9):617–622.

Cho, T., Weiss, R., and Bello, J. (2010). Exploring common variations in state of the art chord recognition systems. In *Proceedings of the Sound and Music Computing Conference (SMC)*.

Choi, A. (2011). Jazz harmonic analysis as optimal tonality segmentation. *Computer Music Journal*, 35(2):49–66.

Chomsky, N. (1957). *Syntactic Structures*. Mouton.

Clarke, E. (1986). Theory, analysis and the psychology of music: A critical evaluation of Lerdahl, F. and Jackendoff, R., A Generative Theory of Tonal Music. *Psychology of Music*, 14(1):3–16.

Cormen, T., Leiserson, C., Rivest, R., and Stein, C. (2001). *Introduction to Algorithms*. MIT press.

Davies, M. and Plumbley, M. (2007). Context-dependent beat tracking of musical audio. *IEEE Transactions on Audio, Speech, and Language Processing*, 15(3):1009–1020.

Deliège, I., Mélen, M., Stammers, D., and Cross, I. (1996). Musical schemata in real time listening to a piece of music. *Music Perception*, 14(2):117–160.

Desain, P. and Honing, H. (1993). Tempo curves considered harmful. *Contemporary Music Review*, 7(2):123–138.

Devriese, D. and Piessens, F. (2011). Explicitly recursive grammar combinators–a better model for shallow parser DSLs. In *proceedings of Practical Aspects of Declarative Languages (PADL)*, pages 84–98. Springer.

Downie, J. (2003). Music information retrieval. *Annual Review of Information Science and Technology*, 37(1):295–340.

Downie, J. (2008). The music information retrieval evaluation exchange (2005–2007): A window into music information retrieval research. *Acoustical Science and Technology*, 29(4):247–255.

Downie, J., Byrd, D., and Crawford, T. (2009). Ten years of ISMIR: Reflections on challenges and opportunities. In *Proceedings of the 10th Society for Music Information Retrieval Conference (ISMIR)*, pages 13–18.

Downie, J., Ehmann, A., Bay, M., and Jones, M. (2010). The music information retrieval evaluation exchange: Some observations and insights. *Advances in Music Information Retrieval*, pages 93–115.

Duxbury, C., Bello, J., Davies, M., and Sandler, M. (2003). Complex domain onset detection for musical signals. In *Proceedings of the Digital Audio Effects Workshop (DAFx)*.

Earley, J. (1970). An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2):94–102.

Ellis, D. (2007). Beat tracking by dynamic programming. *Journal of New Music Research*, 36(1):51–60.

Fujishima, T. (1999). Realtime chord recognition of musical sound: A system using common lisp music. In *Proceedings of the International Computer Music Conference (ICMC)*, pages 464–467.

Gannon, P. (1990). Band-in-a-Box. PG Music.

Gómez, E. (2006). Tonal description of polyphonic audio for music content processing. *INFORMS Journal on Computing*, 18(3):294–304.

Grachten, M., Arcos, J., and Lopez de Mantaras, R. (2004). Melodic similarity: Looking for a good abstraction level. In *Proceedings of the 5th Society for Music Information Retrieval Conference (ISMIR)*, pages 210–215.

Griffiths, P. (accessed January 18, 2012). Varèse, edgard. In *Grove Music Online*. Oxford Music Online, `http://www.oxfordmusiconline.com/subscriber/article/grove/music/29042`.

Gupta, A. and Nishimura, N. (1995). Finding largest common embeddable subtrees. In *Proceedings of the Twelfth Annual Symposium on Theoretical Aspects of Computer Science*, pages 397–408. Springer.

Gupta, A. and Nishimura, N. (1998). Finding largest subtrees and smallest supertrees. *Algorithmica*, 21(2):183–210.

De Haas, W. B., Magalhães, J. P., Veltkamp, R. C., and Wiering, F. (under review 2011a). HarmTrace: Automatic functional harmonic analysis. *Computer Music Journal*.

De Haas, W. B., Magalhães, J. P., Wiering, F., and Veltkamp, R. C. (2011b). HarmTrace: Improving harmonic similarity estimation using functional harmony analysis. In *Proceedings of the 12th International Society for Music Information Retrieval Conference (ISMIR)*.

De Haas, W. B., Robine, M., Hanna, P., Veltkamp, R., and Wiering, F. (2011c). Comparing approaches to the similarity of musical chord sequences. In Ystad, S., Aramaki, M., Kronland-Martinet, R., and Jensen, K., editors, *Exploring Music Contents*, volume 6684 of *Lecture Notes in Computer Science*, pages 242–258. Springer Berlin / Heidelberg.

De Haas, W. B., Rohrmeier, M., Veltkamp, R. C., and Wiering, F. (2009). Modeling harmonic similarity using a generative grammar of tonal harmony. In *Proceedings of the 10th International Society for Music Information Retrieval Conference (ISMIR)*, pages 549–554.

De Haas, W. B., Veltkamp, R. C., and Wiering, F. (2008). Tonal pitch step distance: A similarity measure for chord progressions. In *Proceedings of the 9th International Society for Music Information Retrieval Conference (ISMIR)*, pages 51–56.

De Haas, W. B. and Wiering, F. (2010). Hooked on music information retrieval. *Empirical Musicology Review*, 5(4):176–185.

De Haas, W. B., Wiering, F., and Veltkamp, R. C. (under review 2011d). A geometrical distance measure for determining the similarity of musical harmony. *International Journal of Multimedia Information Retrieval (IJMIR)*.

Hamanaka, M., Hirata, K., and Tojo, S. (2006). Implementing "A Generative Theory of Tonal Music". *Journal of New Music Research*, 35(4):249–277.

Hanna, P., Robine, M., and Rocher, T. (2009). An alignment based system for chord sequence retrieval. In *Proceedings of the 2009 Joint International Conference on Digital Libraries*, pages 101–104. ACM New York, NY, USA.

Harris-Jones, C. and Haines, T. (1998). Sample size and misclassification: Is more always better. In *Proceedings of the Second International Conference on the Practical Application of Knowledge Discovery and Data Mining*.

Harte, C., Sandler, M., Abdallah, S., and Gómez, E. (2005). Symbolic representation of musical chords: A proposed syntax for text annotations. In *Proceedings of the 6th International Society for Music Information Retrieval Conference (ISMIR)*, pages 66–71.

Honing, H. (2010). Lure(d) into listening: The potential of cognition-based music information retrieval. *Empirical Musicology Review*, 5(4):146–151.

Honing, H. and De Haas, W. B. (2008). Swing once more: Relating timing and tempo in expert jazz drumming. *Music Perception: An Interdisciplinary Journal*, 25(5):471–476.

Honing, H. and Ladinig, O. (2009). Exposure influences expressive timing judgments in music. *Journal of Experimental Psychology: Human Perception and Performance*, 35(1):281–288.

Huron, D. (2000). Perceptual and cognitive applications in music information retrieval. In *Proceedings of the 1st International Society for Music Information Retrieval Conference (ISMIR)*, pages 83–92.

Jeuring, J., Leather, S., Magalhães, J. P., and Rodriguez Yakushev, A. (2009). Libraries for generic programming in Haskell. In Koopman, P., Plasmeijer, R., and Swierstra, D., editors, *Advanced Functional Programming (AFP), 6th International School, Revised Lectures*, volume 5832 of *Lecture Notes in Computer Science*, pages 165–229. Springer.

Jiang, N., Grosche, P., Konz, V., and Müller, M. (2011). Analyzing chroma feature types for automated chord recognition. In *Audio Engineering Society Conference: 42nd International Conference: Semantic Audio*.

Kilpeläinen, P. (1992). *Tree Matching Problems with Applications to Structured Text Databases*. PhD thesis, Departement of Computer Science, University of Helsinki.

Klapuri, A. (2005). A perceptually motivated multiple-F0 estimation method. In *Proceedings of IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, pages 291–294.

Koelsch, S. (2011). Toward a neural basis of music perception–a Review and updated model. *Frontiers in Psychology*, 2:1–20.

Kostka, S. and Payne, D. (1984). *Tonal Harmony with an Introduction to 20th-century Music*. McGraw-Hill.

Van Kranenburg, P., Volk, A., Wiering, F., and Veltkamp, R. C. (2009). Musical models for folk-song melody alignment. In *Proceedings of the 10th International Society for Music Information Retrieval Conference (ISMIR)*, pages 507–512.

Krumhansl, C. (2001). *Cognitive Foundations of Musical Pitch*. Oxford University Press, USA.

Krumhansl, C. (2004). The cognition of tonality–as we know it today. *Journal of New Music Research*, 33(3):253–268.

Krumhansl, C. and Kessler, E. (1982). Tracing the dynamic changes in perceived tonal organization in a spatial representation of musical keys. *Psychological Review*, 89(4):334–68.

Lawson, C. and Hanson, R. (1974). *Solving Least Squares Problems*, volume 15. Prentice-Hall.

Lerdahl, F. (2001). *Tonal Pitch Space*. Oxford University Press.

Lerdahl, F. and Jackendoff, R. (1996). *A Generative Theory of Tonal Music*. MIT Press.

Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions, and reversals. *Cybernetics and Control Theory*, 10(8):707–710.

Logan, B. (2000). Mel frequency cepstral coefficients for music modeling. In *Proceedings of the 1th International Society for Music Information Retrieval Conference (ISMIR)*.

Loy, G. (1985). Musicians make a standard: the MIDI phenomenon. *Computer Music Journal*, 9(4):8–26.

Magalhães, J. P. and De Haas, W. B. (2011). Functional modelling of musical harmony: an experience report. In *Proceeding of the 16th ACM SIGPLAN International Conference on Functional Programming (ICFP)*, pages 156–162, New York, NY, USA. ACM.

Magalhães, J. P. and Jeuring, J. (2011). Generic programming for indexed datatypes. In *Proceedings of the 7th ACM SIGPLAN Workshop on Generic Programming*, pages 37–46, New York, NY, USA. ACM.

Manning, C. D., Raghavan, P., and Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA.

Marsden, A. (2010). Schenkerian analysis by computer: A proof of concept. *Journal of New Music Research*, 39(3):269–289.

Marshall, R. L. and Leaver, R. A. (accessed January 17, 2012). Chorale. In *Grove Music Online*. Oxford Music Online, `http://www.oxfordmusiconline.com/subscriber/article/grove/music/05652`.

Martin, S. (accessed January 17, 2012). Pep is an earley parser. `http://www.ling.ohio-state.edu/~scott/`.

Matula, D. W. (1978). Subtree isomorphism in $O(n^{5/2})$. *Algorithmic Aspects of Combinatorics*, 2:91–106.

Mauch, M. (2010). *Automatic Chord Transcription from Audio Using Computational Models of Musical Context*. PhD thesis, Queen Mary University of London.

Mauch, M., Cannam, C., Davies, M., Dixon, S., Harte, C., Kolozali, S., Tidhar, D., and Sandler, M. (2009). Omras2 metadata project 2009. In *Proceedings of the 10th International Conference on Music Information Retrieval (ISMIR)*.

Mauch, M. and Dixon, S. (2010). Approximate note transcription for the improved identification of difficult chords. In *Proceedings of the 11th International Society for Music Information Retrieval Conference (ISMIR 2010)*, pages 135–140.

Mauch, M., Dixon, S., Harte, C., Casey, M., and Fields, B. (2007). Discovering chord idioms through beatles and real book songs. In *Proceedings of the 8th International Conference on Music Information Retrieval (ISMIR)*, pages 255–258.

Müller, M., Kurth, F., and Clausen, M. (2005). Audio matching via chroma-based statistical features. In *Proceedings of the 6th International Conference on Music Information Retrieval (ISMIR)*, pages 288–295.

Narmour, E. (1990). *The Analysis and Cognition of Basic Melodic Structures: The Implication-realization Model*. University of Chicago Press.

Noll, T. and Garbers, J. (2004). Harmonic path analysis. In *Perspectives in Mathematical and Computer-Aided Music Theory*, pages 395–427.

Orio, N. (2006). Music retrieval: A tutorial and review. *Foundations and Trends in Information Retrieval*, 1(1):1–96.

Pachet, F. (1999). Surprising harmonies. *International Journal of Computing Anticipatory Systems*, 4:139–161.

Paiement, J.-F., Eck, D., and Bengio, S. (2005). A probabilistic model for chord progressions. In *Proceedings of the 6th International Conference on Music Information Retrieval (ISMIR)*, pages 312–319, London, UK.

Pardo, B. and Birmingham, W. (2002). Algorithms for chordal analysis. *Computer Music Journal*, 26(2):27–49.

Patel, A. D. (2003). Language, Music, Syntax and the Brain. *Nature Neuroscience*, 6:674–681.

Pearce, M. and Wiggins, G. (2006). Expectation in melody: the influence of context and learning. *Music Perception*, 23(5):377–405.

Peyton Jones, S. (2003). Haskell 98 language and libraries: the revised report. *Journal of Functional Programming*, 13(1):7–255.

Peyton Jones, S., Vytiniotis, D., Weirich, S., and Washburn, G. (2006). Simple unification-based type inference for GADTs. In *Proceeding of the 11th ACM SIGPLAN International Conference on Functional Programming (ICFP)*, pages 50–61. ACM.

Pickens, J. and Crawford, T. (2002). Harmonic models for polyphonic music retrieval. In *Proceedings of the 11th International Conference on Information and Knowledge Management*, pages 430–437. ACM New York, NY, USA.

Piston, W. and DeVoto, M. (1991). *Harmony*. Victor Gollancz.

Pitt, M. and Myung, I. (2002). When a good fit can be bad. *Trends in Cognitive Sciences*, 6(10):421–425.

Rameau, J.-P. (1971). *Treatise on Harmony*. Dover Publications. Translated by Philip Gossett.

Riemann, H. (1893). *Vereinfachte Harmonielehre; oder, die Lehre von den tonalen Funktionen der Akkorde*. Augener.

Roads, C. (1979). Grammars as representations for music. *Computer Music Journal*, 3(1):48–55.

Rohrmeier, M. (2007). A generative grammar approach to diatonic harmonic structure. In Georgaki, Kouroupetroglou, A., editor, *Proceedings of the 4th Sound and Music Computing Conference*, pages 97–100.

Rohrmeier, M. (2011). Towards a generative syntax of tonal harmony. *Journal of Mathematics and Music*, 5(1):35–53.

Rohrmeier, M. and Cross, I. (2008). Statistical properties of tonal harmony in Bach's chorales. In *Proceedings of the 10th International Conference on Music Perception and Cognition (ICMPC)*, pages 619–627.

Schellenberg, E. G. (1996). Expectancy in melody: tests of the implication-realization model. *Cognition*, 58(1):75–125.

Schenker, H. (1935). *Der Freie Satz. Neue musikalische Theorien und Phantasien*. Universal-Edition, Vienna.

Schrijvers, R., Peyton Jones, S., Chakravarty, M. M. T., and Sulzmann, M. (2008). Type checking with open type functions. In *Proceedings of the 13th ACM SIGPLAN International Conference on Functional Programming (ICFP)*, pages 51–62.

Schrijvers, T., Peyton Jones, S., Sulzmann, M., and Vytiniotis, D. (2009). Complete and decidable type inference for GADTs. In *Proceedings of the 14th ACM SIGPLAN International Conference on Functional Programming (ICFP)*, pages 341–352.

Serrà, J., Gómez, E., Herrera, P., and Serra, X. (2008). Chroma binary similarity and local alignment applied to cover song identification. *IEEE Transactions on Audio, Speech, and Language Processing*, 16(6):1138–1151.

Smith, S. (1997). *The Scientist and Engineer's Guide to Digital Signal Processing*. California Technical Publishing, San Diego, California.

Smith, T. F. and Waterman, M. S. (1981). Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197.

Steedman, M. J. (1984). A generative grammar for jazz chord sequences. *Music Perception*, 2(1):52–77.

Steedman, M. J. (1996). The blues and the abstract truth: Music and mental models. In Oakhill, J. and Garnham, A., editors, *Mental Models in Cognitive Science: Essays in Honour of Phil Johnson-Laird*, chapter 15, pages 305–318. Psychology Press.

Swierstra, S. D. (2009). Combinator parsers: a short tutorial. In Bove, A., Barbosa, L., Pardo, A., , and Sousa Pinto, J., editors, *Language Engineering and Rigorous Software Development*, volume 5520 of *Lecture Notes in Computer Science*, pages 252–300. Spinger.

Temperley, D. (2001). *The Cognition of Basic Musical Structures*. Cambridge, MA, MIT Press.

Temperley, D. and Sleator, D. (1999). Modeling meter and harmony: A preference-rule approach. *Computer Music Journal*, 23(1):10–27.

Typke, R. (2007). *Music Retrieval Based on Melodic Similarity*. PhD thesis, Utrecht University.

Various authors (2005). *The Real Book*. Hal Leonard Corporation, 6th edition.

Vincent, E., Bertin, N., and Badeau, R. (2010). Adaptive harmonic spectral decomposition for multiple pitch estimation. *IEEE Transactions on Audio, Speech, and Language Processing*, 18(3):528–537.

Wakefield, G. H. (1999). Mathematical representation of joint time-chroma distributions. In *Part of the Society of Photographic Instrumentation Engineers Conference on Advanced Signal Processing Algorithms, Architectures, and Implementations (SPIE)*, pages 637–645.

Wiering, F. (2009). Meaningful music retrieval. In *1st Workshop on the Future of Music–Proceedings of the 10th International Society for Music Information Retrieval Conference (ISMIR)*.

Wiering, F., de Nooijer, J., Volk, A., and Tabachneck-Schijf, H. (2009a). Cognition-based segmentation for music information retrieval systems. *Journal of New Music Research*, 38(2):139–154.

Wiering, F., Veltkamp, R. C., Garbers, J., Volk, A., van Kranenburg, P., and Grijp, L. P. (2009b). Modelling folksong melodies. *Interdisciplinary Science Reviews*, 34(2-3):154–171.

Wiggins, G. (2009). Semantic gap?? schemantic schmap!! methodological considerations in the scientific study of music. In *Proceedings of the 11th IEEE International Symposium on Multimedia*, pages 477–482.

Wiggins, G., Mullensiefen, D., and Pearce, M. (2010). On the non-existence of music: Why music theory is a figment of the imagination. In *Musicæ Scientiæ, Discussion Forum*, volume 5, pages 231–255. ESCOM.

Winograd, T. (1968). Linguistics and the computer analysis of tonal harmony. *Journal of Music Theory*, 12(1):2–49.

Wolff, C., Emery, W., Wollny, P., Leisinger, U., and Roe, S. (accessed January 17, 2011). Bach. In *Grove Music Online*. Oxford Music Online, `http://www.oxfordmusiconline.com/subscriber/article/grove/music/40023pg10`.

Appendix: HarmTrace harmony model

```
module HarmTrace.Models.Jazz.Model where


#ifndef η
#define η (Su (Su (Su (Su (Su Ze)))))
#endif


data Maj_Mode
data Min_Mode


  -- High level structure parametrised by mode μ
data Piece = forall μ ∘ Piece [Func μ]


  -- The Functional level
data Func μ where
  PT :: Ton μ  → Func μ
  PD :: Dom μ → Func μ


  -- Harmonic categories:
  -- Tonic
data Ton μ where
    -- major mode
  T_1    :: Final I Maj          → Ton Maj_Mode
  T_2    :: Final I Maj          → Final IV Maj → Final I Maj → Ton Maj_Mode
  T_3^par :: Final III Min       → Ton Maj_Mode
  T_4^par :: Ton_m^bor           → Ton Maj_Mode
    -- minor mode
  Tm_1   :: SD Min_Mode I Min → Ton Min_Mode
  Tm_2   :: Final I Min          → Final IV Min → Final I Min → Ton Min_Mode
```

```
  Tm_3^par :: Final III♭ Maj        → Ton Min_Mode
  Tm_4^par :: Ton^bor               → Ton Min_Mode


  -- Dominant
data Dom μ where
    -- major mode
  D_1      :: SDom μ → Dom μ        → Dom μ
  D_2      :: SD μ V Dom            → Dom μ
  D_3      :: SD μ V Maj            → Dom μ
  D_4      :: SD Maj_Mode VII Min   → Dom Maj_Mode
  D_5^bor  :: Dom_m^bor             → Dom Maj_Mode
    -- minor mode
  Dm_4     :: SD Min_Mode VII♭ Maj  → Dom Min_Mode
  Dm_5^bor :: Dom^bor               → Dom Min_Mode


  -- Sub-dominant
data SDom μ where
  S_1^par  :: SD μ II Min           → SDom μ
  S_2^par  :: SD μ II Dom           → Final II Min  → SDom μ
  S_3      :: SD Maj_Mode IV Maj    → SDom Maj_Mode
  S_4      :: SD Maj_Mode III Min   → Final IV Maj → SDom Maj_Mode
    -- borrowing from minor in a major mode
  S_5^bor  :: SDom_m^bor            → SDom Maj_Mode
    -- minor mode
  Sm_3     :: SD Min_Mode IV Min    → SDom Min_Mode
  Sm_4     :: SD Min_Mode III♭ Maj  → Final IV Min → SDom Min_Mode
  Sm_5^bor :: SDom^bor              → SDom Min_Mode
  Sm_6     :: SD Min_Mode II♭ Maj   → SDom Min_Mode   -- Neapolitan


  -- Borrowings from minor in a major
data Ton_m^bor  = Tm_21^bor (SD Min_Mode I    Min)
                | Tm_23^bor (SD Min_Mode III♭ Maj)
data Dom_m^bor  = Dm_24^bor (SD Min_Mode VII♭ Maj)
data SDom_m^bor = Sm_20^bor (SD Min_Mode IV   Min)
                | Sm_22^bor (SD Min_Mode II♭  Maj)   -- Neapolitan
```

```
  -- Borrowings from major in a minor
data Ton^bor    = T_21^bor (SD Maj_Mode I   Maj)
                | T_23^bor (SD Maj_Mode III Min)
data Dom^bor    = D_24^bor (SD Maj_Mode VII Min)
data SDom^bor   = S_20^bor (SD Maj_Mode IV  Maj)


  -- Limit secondary Dominants to η recursive levels
  -- δ denotes the scale degree
  -- μ denotes the mode
  -- γ denotes the chord class
type SD μ δ γ = Base_sd δ γ η


  -- A scale degree that can optionally substituted by its tritone substitution
  -- or transformed in a diminished seventh chord (7♭9)
type Tritsub δ γ = Base_final δ γ (Su (Su Ze))


  -- A scale degree that can only translate into a surface chord,
  -- but in case of a diminished seventh chord, it allows for the transformation
  -- into enharmonic equivalent diminished seventh surface chords
type DimTrans δ γ = Chord δ γ (Su (Su (Su (Su Ze))))


  -- A scale degree that translates into a (non-transformable) surface chord
type Final δ γ = Chord δ γ (Su Ze)


data Base_sd δ γ η where
  Base_sd      :: Tritsub δ γ                → Base_sd δ γ (Su η)
    -- Rule for explaining secondary dominants
  Cons_VDom    :: Base_sd (V/ δ) Dom η       → Base_sd δ γ η      → Base_sd δ γ (Su η)
  Cons_Diat    :: Base_sd (Vd/ δ) Min η      → Base_sd δ Min η    → Base_sd δ Min (Su η)
  Cons_DiatM   :: Base_sd (Vd^m/ δ) Maj η    → Base_sd δ Maj η    → Base_sd δ Maj (Su η)
  Cons'_DiatM  :: Base_sd (Vd^m/ δ) Maj η    → Base_sd δ Min η    → Base_sd δ Min (Su η)
    -- Minor fifth insertion
  Cons_Vmin    :: Base_sd (v/ δ) Min η       → Base_sd δ Dom η    → Base_sd δ Dom (Su η)


data Base_final δ γ η where
    -- Just a “normal", Final degree.
```

```
  Base_final     :: DimTrans δ γ              → Base_final δ γ (Su η)
    -- Tritone substitution
  Cons_tritone   :: Base_final (II♭/ δ) Dom η  → Base_final δ Dom (Su η)
  Cons_dim       :: Base_final (II♭⁰/ δ) Dim η → Base_final δ Dom (Su η)

  -- Diminished tritone substitution accounting for diminished chord transitions
data Chord δ γ η where
  Chord          :: ChordToken          → Chord δ γ (Su η)
  Cons_transpose :: Chord (III♭/ δ) Dim η → Chord δ Dim (Su η)

  -- type-level chord classes
data Maj
data Min
data Dom
data Dim

  -- Degrees (at the type level)
data I;   data I♭;  data I♯;  data II;  data II♭;  data II♯;  data III;
data III♭; data III♯; data IV; data IV♭; data IV♯; data V;  data V♭;
data V♯; data VI; data VI♭; data VI♯; data VII; data VII♭; data VII♯

  -- Used when we don't want to consider certain derivations
data Imp

  -- Type to value conversions
class ToClass γ where
  toClass :: γ → ClassType

instance ToClass Maj  where toClass _ = Maj
instance ToClass Min  where toClass _ = Min
instance ToClass Dom where toClass _ = Dom
instance ToClass Dim  where toClass _ = Dim

  -- The class doesn't really matter, since the degree will be impossible to parse
instance ToClass Imp where toClass _ = Dim

class ToDegree δ where
  toDegree :: δ → ScaleDegree
```

```
  -- Note that in Section 4.3 we used Int values
  -- for simplicity. In reality we use a slightly different Note datatype to
  -- represent a value-level scale degree.
instance ToDegree I    where toDegree _ = Note Nothing I
instance ToDegree II   where toDegree _ = Note Nothing II
instance ToDegree III  where toDegree _ = Note Nothing III
instance ToDegree IV   where toDegree _ = Note Nothing IV
instance ToDegree V    where toDegree _ = Note Nothing V
instance ToDegree VI   where toDegree _ = Note Nothing VI
instance ToDegree VII  where toDegree _ = Note Nothing VII
instance ToDegree I♭   where toDegree _ = Note (Just Fl) I
instance ToDegree II♭  where toDegree _ = Note (Just Fl) II
instance ToDegree III♭ where toDegree _ = Note (Just Fl) III
instance ToDegree IV♭  where toDegree _ = Note (Just Fl) IV
instance ToDegree V♭   where toDegree _ = Note (Just Fl) V
instance ToDegree VI♭  where toDegree _ = Note (Just Fl) VI
instance ToDegree VII♭ where toDegree _ = Note (Just Fl) VII
instance ToDegree II♯  where toDegree _ = Note (Just Sh) II
instance ToDegree III♯ where toDegree _ = Note (Just Sh) III
instance ToDegree IV♯  where toDegree _ = Note (Just Sh) IV
instance ToDegree V♯   where toDegree _ = Note (Just Sh) V
instance ToDegree VI♯  where toDegree _ = Note (Just Sh) VI
instance ToDegree VII♯ where toDegree _ = Note (Just Sh) VII
  -- Can't ever parse these
instance ToDegree Imp where toDegree _ = Note Nothing Imp

  -- Diatonic fifths, and their class
  -- (in the comments we use the C Major scale as a reference)
type family Vd/ δ :: ⋆
type instance Vd/ I    = Imp  -- V – G^7 preferred to be analysed as Dom
type instance Vd/ V    = Imp  -- Dm^7 – preferred to be analysed as SDom
type instance Vd/ II   = VI   -- Ab
type instance Vd/ VI   = III  -- Em^7
type instance Vd/ III  = VII  -- Bm7♭5 – can be explained by Cons_dim specification
type instance Vd/ VII  = Imp  -- IV – FMaj7 preferred to be analysed as SDom
type instance Vd/ IV   = Imp  -- I – CMaj7
type instance Vd/ II♭  = Imp
type instance Vd/ III♭ = Imp
type instance Vd/ IV♯  = Imp
type instance Vd/ VI♭  = Imp
```

```
type instance Vd/VII♭ = Imp
type instance Vd/Imp = Imp


  -- Diatonic fifth derivation in minor
type family Vd^m/δ :: ⋆
type instance Vd^m/I    = Imp
type instance Vd^m/V    = Imp
type instance Vd^m/II   = VI♭
type instance Vd^m/VI   = Imp
type instance Vd^m/III  = Imp
type instance Vd^m/VII  = Imp
type instance Vd^m/IV   = Imp
type instance Vd^m/II♭  = Imp
type instance Vd^m/III♭ = VII♭
type instance Vd^m/IV♯  = Imp
type instance Vd^m/VI♭  = III♭
type instance Vd^m/VII♭ = Imp
type instance Vd^m/Imp = Imp


  -- Type families for secondary Dominants
  -- Perfect fifths (the class is always Dominant)
type family V/δ :: ⋆
type instance V/I    = Imp   -- interferes with Dom
type instance V/II♭  = VI♭
type instance V/II   = VI
type instance V/III♭ = VII♭
type instance V/III  = VII
type instance V/IV   = I
type instance V/IV♯  = II♭
type instance V/V    = II
type instance V/VI♭  = III♭
type instance V/VI   = III
type instance V/VII♭ = IV
type instance V/VII  = IV♯
type instance V/Imp = Imp
```

```
  -- Perfect fifths for the minor case (this is an additional
  -- type family to control the reduction of ambiguities
  -- specifically in the minor case)
type family v/ δ :: ⋆
type instance v/ I     = V
type instance v/ II♭   = VI♭
type instance v/ II    = VI
type instance v/ III♭  = VII♭
type instance v/ III   = VII
type instance v/ IV    = I
type instance v/ IV♯   = II♭
type instance v/ V     = Imp   -- II – interferes with SDom
type instance v/ VI♭   = III♭
type instance v/ VI    = III
type instance v/ VII♭  = Imp   -- IV – interferes with SDom
type instance v/ VII   = IV♯
type instance v/ Imp   = Imp
```

```
  -- The tritone substitution
type family II♭/ δ :: ⋆
type instance II♭/ I     = IV♯
type instance II♭/ IV♯   = I
type instance II♭/ II♭   = V
type instance II♭/ V     = II♭
type instance II♭/ II    = VI♭
type instance II♭/ VI♭   = II
type instance II♭/ III♭  = VI
type instance II♭/ VI    = III♭
type instance II♭/ III   = VII♭
type instance II♭/ VII♭  = III
type instance II♭/ IV    = VII
type instance II♭/ VII   = IV
type instance II♭/ Imp   = Imp
```

```
  -- Type families for diminished chord transformations
  -- in combination with the secondary dominants and enharmonic equivalence
  -- these type families account for ascending dim chord progressions
type family II♭0/δ :: ⋆
type instance II♭0/I    = II♭
type instance II♭0/II♭  = II
type instance II♭0/II   = III♭
type instance II♭0/III♭ = III
type instance II♭0/III  = IV
type instance II♭0/IV   = IV♯
type instance II♭0/IV♯  = V
type instance II♭0/V    = VI♭
type instance II♭0/VI♭  = VI
type instance II♭0/VI   = VII♭
type instance II♭0/VII♭ = VII
type instance II♭0/VII  = I
type instance II♭0/Imp = Imp


  -- Dim chords can be transposed a minor third without changing their role,
  -- they are enharmonically equivalent.
type family III♭/δ :: ⋆
type instance III♭/I    = III♭
type instance III♭/II♭  = III
type instance III♭/II   = IV
type instance III♭/III♭ = IV♯
type instance III♭/III  = V
type instance III♭/IV   = VI♭
type instance III♭/IV♯  = VI
type instance III♭/V    = VII♭
type instance III♭/VI♭  = VII
type instance III♭/VI   = I
type instance III♭/VII♭ = II♭
type instance III♭/VII  = II
type instance III♭/Imp = Imp
```

Samenvatting

MUZIEK is tegenwoordig overal om ons heen aanwezig: we luisteren naar muziek via onze draagbare mp3-spelers, onze mobiele telefoon, internet en niet te vergeten via de radio. Niet alleen hebben wij hierdoor bijna overal en op ieder moment toegang tot muziek, er is ook een overweldigend aanbod aan muziek waaruit we kunnen kiezen. Het aantal albums dat op dit moment wordt aangeboden door populaire internet diensten als Spotify, iTunes of Last.fm is een veelvoud van het aantal compact discs en langspeelplaten dat vroeger verkrijgbaar was bij de platenzaak om de hoek. Om deze enorme collecties digitale muziek te ordenen en toegankelijk te houden, kunnen muziek liefhebbers wel wat hulp gebruiken; het is immers een ondoenlijke taak om al deze liedjes te beluisteren, en met de hand te annoteren en rangschikken. Daar komt bij dat een aanzienlijk deel van de muziek, die beschikbaar is op het internet, niet of slecht voorzien is van digitale etiketten die de in muzikale inhoud beschrijven. De gebruikers die zo vriendelijk zijn geweest hun muzikale ervaringen met u te delen, zijn soms in de haast vergeten te vermelden wat ze precies met u wilden delen. Denk hierbij bijvoorbeeld aan opnames van een concert gemaakt met een mobiele telefoon, of oude televisieopnames op Youtube.

Zoeken naar muziek op basis van tonale harmonie

In het proefschrift dat u in uw hand houdt, onderzoeken wij verschillende methoden en technieken die ons kunnen helpen met het ordenen van muziek zonder dat er speciale digitale etiketten nodig zijn die de muzikale informatie beschrijven. Onze methoden kijken louter naar de muzikale informatie zoals deze wordt aangetroffen op een CD, in digitale bladmuziek of op een andere digitale informatiedrager. Hiermee behoort dit proefschrift tot het vakgebied dat zich richt op het content-gebaseerd ontsluiten van muzikale informatie; in het Engels wordt dit vakgebied, dat onderdeel is van de discipline informatica, Music Information Retrieval (MIR) genoemd. Dit proefschrift bestaat uit zeven hoofdstukken waarin verschillende onderwerpen de revue passeren, maar waarin de gelijkenis voor muzikale documenten (muziekstukken, bladmuziek, akkoordenschema's of andere muzikale representaties) een hoofdrol vervult.

Dat muzikale gelijkenis centraal staat in het MIR vakgebied is niet zo verwonderlijk. Gelijkenismaten maken het mogelijk om een collectie van documenten te rangschikken op basis van de berekende gelijkenis van deze documenten. Uiteraard zijn er vele verschillende soorten muzikale gelijkenis denkbaar, men zou bijvoorbeeld een muziekstuk kunnen vergelijken op basis van de melodie, op basis van het genre van het stuk, op basis van de instrumenten die erin voorkomen of op basis van het tempo van het stuk. Wanneer we weten in hoeverre muziekstukken in deze aspecten van elkaar verschillen, zouden we volautomatisch een zoekvraag kunnen beantwoorden als: “Mag ik van u een rustig jazz nummer waarin de trompet de melodie speelt?”. Helaas is een systeem dat dit kan op het moment van schrijven nog toekomstmuziek.

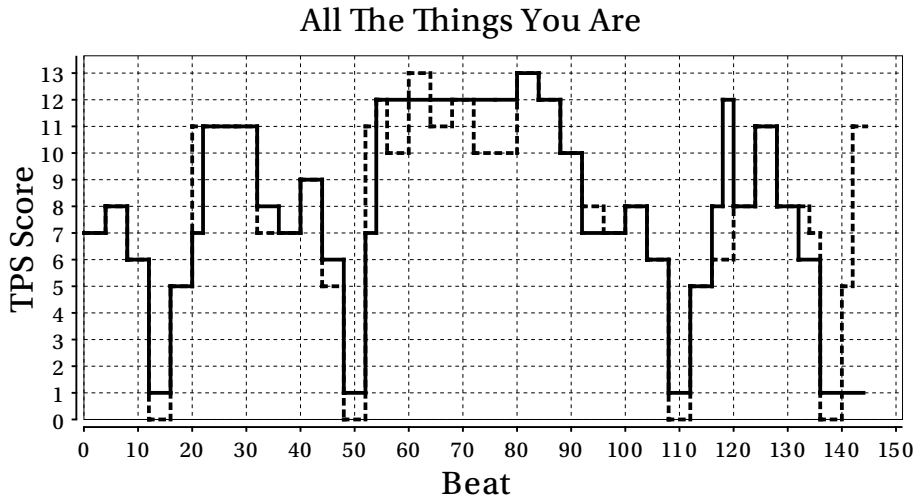
In dit proefschrift richten we ons op het modelleren en definiëren van een specifieke soort muzikale gelijkenis: harmonische gelijkenis. Harmonie ontstaat in muziek wanneer er meer dan twee noten tegelijkertijd klinken. Deze samenklinkende noten vormen akkoorden, die op hun beurt weer gebruikt kunnen worden om de harmonische structuur van een muziek stuk over de tijd te representeren. De klassieke muziektheorie leert ons dat er veel regelmaat te vinden is in de manier waarop akkoorden worden gemaakt en worden gecombineerd tot akkoordenschema’s. Sommige combinaties van akkoorden klinken natuurlijk, terwijl andere combinaties vreemd en tegennatuurlijk klinken. Dit kan liggen aan de interne structuur van het akkoord, dat wil zeggen de relatieve afstanden tussen de noten waaruit het akkoord is opgebouwd, of aan de combinatie van de akkoorden in het stuk. Een belangrijk concept binnen de muziektheorie is toonsoort. De toonsoort bepaalt de grondtoon van een muziekstuk en bepaald ook de verzameling noten die relatief goed samen klinken. Vandaar dat de relatie tussen een akkoord en de toonsoort veel informatie geeft over de *harmonische functie* van het akkoord. Natuurlijk is er over harmonieleer veel meer te vertellen; er zijn niet voor niets de afgelopen 150 jaar vele boeken over geschreven.

Een gelijkenismaat op basis van muzikale harmonie is nuttig voor het beantwoorden van verschillende zoekvragen. Zo kan een dergelijke maat gebruikt worden om zogeheten *cover-songs*—een niet-originele uitvoering van een stuk, vaak uitgevoerd door een andere artiest—op te sporen. In cover-songs kan de uitvoering, de stijl en soms ook de melodie variëren, maar de harmonie blijft over het algemeen stabiel. Ook maakt een maat van harmonische gelijkenis het mogelijk op bepaalde families van muziekstukken te zoeken, denk hierbij bijvoorbeeld aan de familie van bluesliederen, maar ook aan de Goldbergvariaties van Bach. Behalve aan harmonische gelijkenismaten wordt er in dit proefschrift ook uitgebreid aandacht besteed aan automatische harmonische analyse van akkoordenschema’s en aan het automatisch afleiden van akkoordsymbolen uit het ruwe audio signaal.

Overzicht van hoofdstukken

We beginnen het proefschrift in hoofdstuk 1 met het uiteenzetten van de kaders van het in dit proefschrift gepresenteerde onderzoek door het introduceren van het MIR vakgebied, het muzikale gelijkenisprincipe en enige basale harmonieleer. Vervolgens beargumenteren we waarom modellen van muzikale kennis noodzakelijk zijn voor het ontwerpen van betekenisvolle muzikale gelijkenis en het oplossen van aanverwante taken. Tegenwoordig worden veel typische MIR taken opgelost met generieke computermodellen, die de statistische distributies leren uit muzikale data. Heel simpel gezegd houdt dit in dat men modellen afleidt door te tellen hoe vaak een bepaalde karakteristieke eigenschappen van de muziek zich voor doen in een grote verzameling liedjes. Een hierbij, in onze optiek, veel gemaakte fout is de aanname dat alle informatie die nodig is voor het kunnen definiëren van zinvolle muzikale gelijkenis in de data kan worden gevonden. Wij achten dit hoogst onwaarschijnlijk, mede doordat onderzoek heeft aangetoond dat mensen veel bewuste en onbewuste kennis over muziek hebben die niet direct af te leiden is uit de muzikale informatie. Voor betekenisvolle muzikale gelijkenis moet deze kennis dus gemodelleerd en toegepast worden. Deze twee zaken, het modelleren van muzikale kennis en het toepassen ervan, vormen de kern van dit proefschrift.

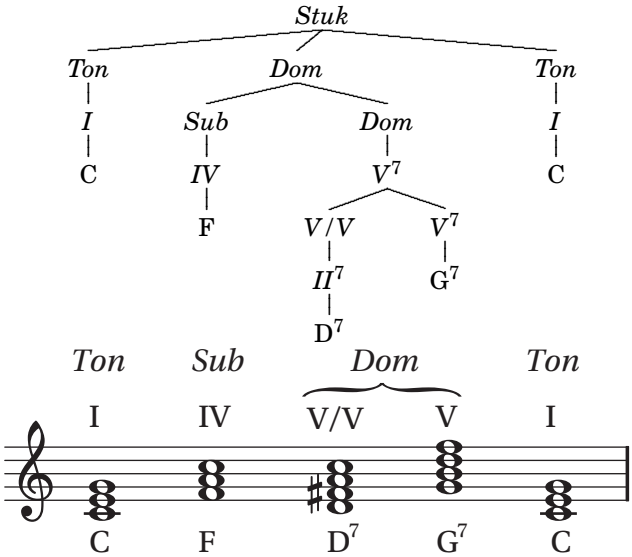
In hoofdstuk 2 introduceren we een eerste harmonische gelijkenismaat, de *Tonal Pitch Step Distance* (TPSD). We nemen aan dat we de akkoordsymbolen van een muziekstuk tot onze beschikking hebben en aan de hand van het *Tonal Pitch Space* (TPS) model van Fred Lerdahl bepalen we hoeveel deze op elkaar lijken. Het TPS model kan worden gebruikt om de perceptuele afstand tussen twee willekeurige akkoorden te schatten. Vervolgens gebruiken we dit model om, voor een gegeven akkoordenschema, de afstanden tussen het akkoord en de toonsoort te bepalen. Wanneer je deze afstanden weergeeft over de tijd ontstaat er een stapsgewijs constante functie. De afstand tussen twee van zulke functies kan worden gedefinieerd als de totale oppervlakte tussen de twee functies, geminimaliseerd over alle mogelijk horizontale verschuivingen, zie bijvoorbeeld figuur 7.1. We verkrijgen de uiteindelijke afstandsmaat door deze minimale oppervlakte te delen door de lengte van het kortste stuk. Hierdoor normaliseren we de afstandsmaat en leiden korte akkoordenschema's niet altijd tot een kleinere afstand. Vervolgens tonen we aan dat deze harmonische afstandsmaat effectief gebruikt kan worden om verschillende akkoordenschema's van hetzelfde jazzstuk te zoeken en dat hierbij de drieklank informatie in het akkoord het belangrijkste is voor de bepalen van de harmonische gelijkheid. Daarnaast laten we zien dat de TPSD ook nieuwe musicologische inzichten kan opleveren in de relatie tussen de harmonie en melodie in Bach's koralen.



Figuur 7.1: Een diagram dat de vergelijking tussen twee verschillende versies van de jazz standard *All the things you are* weergeeft. De totale oppervlakte tussen de twee stapsgewijs constante functies, genormaliseerd met de lengte van het kortste stuk, definieert de afstand tussen de twee stukken. De minimale afstand wordt gevonden door een van de functies cyclisch te verschuiven.

In hoofdstuk 3 benaderen we harmonische gelijkheid vanuit een ander perspectief dat vergelijkbaar is met hoe muzikanten akkoordenschema's behandelen. Wanneer een muzikant een akkoordenschema bestudeert, kijkt hij of zij niet alleen naar het akkoord, maar ook naar de context waarin het akkoord zich bevindt. Zowel de globale context, de toonsoort, evenals de locale context, de omringende akkoorden, beïnvloeden hoe een akkoord wordt waargenomen. In dit hoofdstuk presenteren we een eerste methode om automatisch de harmonische functie van een akkoord te herleiden. Een belangrijke aanname die we hierbij doen is dat muzikale harmonie hiërarchisch geordend is. Een belangrijk gegeven dat duidt op een duidelijk hiërarchische ordening is het feit dat niet ieder akkoord even belangrijk is. Sommige akkoorden kunnen weg worden gelaten zonder het verdere akkoordenschema in de war te sturen, terwijl als bepaalde belangrijke akkoorden worden weggelaten, slechts een onsamenhangend akkoordenschema overblijft.

Als basis voor de automatische harmonische analyse modelleren we de regels van de westerse tonale harmonieleer met een Chomskyaanse grammatica, vergelijkbaar met de grammatica's die jarenlang in de natuurlijke taalverwerking zijn gebruikt. De regels van deze grammatica specificeren welke akkoordopeenvolgingen logisch zijn volgens de harmonieleer en welke niet. Door vervolgens de akkoordenschema's te *parsen*,



Figuur 7.2: Een veel voorkomende akkoordopeenvolging en de harmonische analyse zoals deze gegenereerd wordt door HARMTRACE. De akkoordsymbolen zijn weergegeven onder en de harmonische analyse boven de notenbalk. Om het voorbeeld eenvoudig te houden, hebben we geen gebruikelijke stemvoeringen toegevoegd.

het automatisch uitrekenen van de hiërarchische verbanden tussen akkoorden gegeven een grammatica, kunnen we van ieder akkoord bepalen wat de hiërarchische functie van dat akkoord is in de tonale context, zie figuur 7.2 voor een simpel voorbeeld.

Voor muzikanten en musicologen is het interessant om automatisch een akkoordenschema te kunnen analyseren, met name als grote collecties geanalyseerd moeten worden. In hoofdstuk 4 laten we echter ook zien dat de informatie over de functie van een akkoord ook nuttig kan zijn voor het verbeteren van harmonische gelijkenismaten. De harmonische analyse die ons model oplevert heeft een boomstructuur en in dit hoofdstuk presenteren we een algoritme waarmee we kunnen uitrekenen in hoeverre twee van deze bomen met elkaar overeenkomen. De uiteindelijke gelijkenismaten verkrijgen we door verschillende eigenschappen van de berekende relatie tussen twee bomen te analyseren. Bijvoorbeeld, als er in beide bomen veel gelijke knopen voorkomen, is dit een indicatie dat de akkoordenschema's op elkaar lijken. We laten zien dat deze gelijkenismaten goede resultaten opleveren, maar we signaleren ook een aantal problemen met betrekking tot het parsen van akkoordenschema's evenals het uitrekenen van de relaties tussen bomen. Deze problemen zullen we in respectievelijk hoofdstuk 4 en hoofdstuk 5 uitgebreid behandelen.

In hoofdstuk 4 werken we onze formalisatie van de westerse tonale harmonie verder uit en presenteren we een systeem dat snel en accuraat een harmonische analyse kan doen van een akkoordenschema. Daarnaast laten we zien hoe de nadelen van context-vrij parsen kunnen worden verholpen door het gebruiken van geavanceerde functionele programmeertechnieken die voorhanden zijn in de programmeertaal Haskell. Een lastig probleem van context-vrij parsen is dat, als een akkoordenschema niet precies voldoet aan de specificatie van de grammatica, de parser dan geen analyse kan geven van het akkoordenschema. Dit hebben we opgelost door een parser te gebruiken die in staat is automatisch fouten te corrigeren. Wanneer een akkoordenschema een vreemd akkoord bevat, wordt deze automatisch verwijderd of wordt er een akkoord toegevoegd om het schema aan de regels van de harmonieleer te laten voldoen. Het resultaat van deze inspanningen hebben we HARMTRACE genoemd en we laten zien, aan de hand van een aantal voorbeelden, dat de analyses van HARMTRACE overeenkomen met de traditionele harmonieleer (zie figuur 4.3 tot en met 4.7 op pagina 76 tot en met 79). Verder laten we zien dat HARMTRACE snel grote collecties van akkoordenschema's kan analyseren, ook als deze schema's fouten bevatten.

hoofdstuk 5 hervat de discussie over harmonische gelijkenis van akkoordenschema's en harmonische analyses waar we in hoofdstuk 3 aan begonnen waren. De ideeën over het vergelijken van analysebomen wordt verder uitgewerkt in twee nieuwe gelijkenismaten. Ook introduceren we twee gelijkenismaten die gebaseerd zijn op een principe uit de computationele biologie: het lokaal uitlijnen van reeksen (in het Engels wordt dit *local alignment* genoemd). In één van deze uitlijnprocedures gebruiken we de harmonische annotaties van HARMTRACE en in de andere doen we dat niet. We vergelijken alle maten uit dit hoofdstuk tezamen met de maten uit hoofdstuk 2 in een experiment waarin we wederom verschillende versies van dezelfde jazz standard proberen te vinden. De resultaten laten zien dat de methode die op het lokaal uitlijnen van door HARMTRACE geannoteerde akkoordenschema's is gebaseerd, de beste resultaten oplevert.

De gelijkenismaten die in hoofdstuk 2 en hoofdstuk 5 zijn gepresenteerd, gaan er vanuit dat de gebruiker beschikking heeft over de te vergelijken akkoordenschema's. Dit is uiteraard niet altijd het geval en daarom introduceren wij in hoofdstuk 6 MPTREE: een nieuwe methode om automatisch de akkoorden te onttrekken uit een digitaal audiobestand. We gebruiken bestaande software om uit het ruwe muzikale signaal pulsen spectruminformatie te onttrekken en hiermee kunnen we lijst van op de tel gesynchroniseerde kandidaatakkoorden maken. Wanneer er veel zekerheid is in het signaal over het klinkende akkoord, dan kiezen we het best passende akkoord; wanneer er veel onzekerheid is in het signaal over het klinkende akkoord, dan laten we het HARMTRACE-model kiezen welk van de kandidaatakkoorden het beste past volgens de regels

van de harmonieleer. We evalueren de prestaties van vier verschillende varianten van het MPTrEE-systeem, om de effecten van toonsoortinformatie, het groeperen van akkoorden en het HARMTRACE-model in kaart te brengen. De resultaten laten zien dat de HARMTRACE-variant het beste functioneert.

We sluiten het proefschrift af in hoofdstuk 7 door terug te blikken op de behaalde resultaten en vooruit te kijken naar de mogelijk toekomstig onderzoek dat de gepresenteerde methoden verder zou kunnen verbeteren. Dit proefschrift heeft een aantal waardevolle inzichten opgeleverd met betrekking tot muzikale gelijkenis, automatische harmonische analyse en harmonische gelijkenis in het bijzonder. We zijn daarom hoopvol gestemd dat we, in de nabije toekomst, concrete systemen of applicaties zullen tegenkomen waarin het afleiden, analyseren en vergelijken van akkoordenschema's wordt ingezet voor het ontsluiten van grote collecties digitale muziek.

Curriculum vitae

WILLEM Bas de Haas was born on 4 august 1980 in Groningen. He finished his pre-university education by obtaining his VWO diploma on the subjects Dutch, English, Mathematics, Physics, Chemistry, Biology, Philosophy, and Music at the Noorderpoort College in Groningen. In 2000, he started studying Cognitive Artificial Intelligence at Utrecht University. In 2002, Bas got admitted to the preparatory year of the conservatoire in Utrecht. One year later, this lead to his admission to the jazz guitar training at the same institution. However, Bas did not complete the training, and stopped his jazz guitar education at the conservatoire in 2004. In 2006, he did an internship at the Music Cognition Group at the University of Amsterdam. In 2007, Bas received his Master Degree in Cognitive Artificial Intelligence on the thesis *The role of tempo in groove and swing timing*, which was written also under supervision of Prof.dr. Henkjan Honing. In the same year, he started with his PhD research at Utrecht University leading to his PhD thesis in 2011. Bas continues his Music Information Retrieval research as a post-doctoral researcher at Utrecht University.

Acknowledgements

FINALLY, my thesis is finished! It took me exactly four years and three months to learn many new things, to perform research, and to write it down in a more or less coherent manner. Although it might seem as if writing a PhD thesis is something you do on your own, in my experience, this is clearly not the case. A large number of people were directly or indirectly involved in the realisation of this thesis. Although I experienced doing a PhD as very pleasant in general, there were some moments that I found difficult. Especially at those moments, some of these people were crucial for the successful completion of the thesis you are now holding. I would like to use the last pages of this thesis to thank all of you.

First of all, I would like to thank my supervisors Frans Wiering and Remco Veltkamp. Having both a musicologist and a computer scientist as supervisor, created an excellent basis for studying Music Information Retrieval, and I could not have wished for two more devoted teachers. Frans, I have always enjoyed all of our inspiring conversations of both scientific and non-scientific nature, and I hope there are many more to come. Also, your directions and proof readings have been essential to this thesis. Remco, I admire your clear mind, and your ability to always keep track of the bigger picture. As the head of our group, you have been extremely busy, but you have always made time for me whenever I needed it; this is something I greatly appreciate.

During my PhD research I collaborated with some fellow PhD students. It was a typical rainy day in the early spring of 2010 when I had to present my work at the departmental research day. Afterwards, José Pedro Magalhães approached me and said that he liked my work, but he knew a better way of doing it. Unlike most other people that tell you they know a better way of doing whatever you are doing, Pedro actually did it better. A couple of months later he got back to me, and showed me a prototypical implementation of a harmony model in Haskell. This marked the beginning of a very fruitful collaboration that yielded several papers. Another collaborator that has been important for the work on automatic harmonic analysis is Martin Rohrmeier. Martin, I enjoyed all our stimulating discussions and modelling sessions.

It sometimes takes a PhD thesis to realise again how important the people close to you

really are. A special word of gratitude goes out the love of my life, Mira. I know it was not easy: living with me in the same apartment while I had to finish my thesis. Nonetheless, during all of my grumpy mornings, ill-tempered evenings, and stressful nights, you were there for me. Your love and support (and delicious cooking!) mean everything to me.

I am greatly indebted to my parents. Willem and Pauline, the love and care you gave me, made me into who I am now; thank you. Maaike, zus, thanks for being such a special and independent mind. Joachim, we are so alike and so different, thanks for being such a good friend. Robbert, relaxing, playing Carassonne, and organising LANetten has been priceless. I am proud to have both of you as a *paranimf*.

Playing guitar in several bands has played an important role in my life and has resulted in some of the most valuable friendships I cherish today. Leon, making and producing music with you has been an exceptionally enlightening and delightful experience. Another special thanks goes out to all current and past members of Corvus (a special kind of music): Arnoud, Marcus, Menno, Jan, Frank and Gijs, jongens, bedankt! Gijs, not only have you been a good friend and band member, we have done several pleasant, sometimes prestigious, and usually profitable projects together, which I have always enjoyed. I also want to thank you for designing a wonderful cover and defense invitation. To Jeffrey, Tijmen, and Jornt of B-wax I would like to say: thanks for all the music, and always coming off second best.

I would also like to thank the colleagues of the WITCHCRAFT project. Peter van Kranenburg's annotations have been important for the results in Chapter 2, and Anja Volk's proof reading improved some of the text in this thesis. Furthermore, the *Department of Information and Computing Sciences* and in particular the *Multimedia and Geometry* group has been a very pleasant environment the past four years. Geert-jan, Marc, Ben, Thijs, Reinier, Tom, Saskia, Rodrigo, Maarten, Anne, Marcelo, Xinghan, Frank, Frank, and Frank, thanks for giving me a great time. Finally, a word of thanks goes to all members of the reading committee: dr. Meinard Müller, prof.dr. Geraint Wiggins, prof.dr. Henkjan Honing, prof.dr. Louis Grijp, and prof.dr. Doaitse Swierstra.

A last, but very special, word of gratitude goes out to my cat Sammie. During the final long weekends of thesis writing, he has laid on my desk and kept me company. Gently sitting on my keyboard, whenever I should think instead of write, and subtly stretching himself in front of my monitor whenever I needed rest.

I would like to end by repeating one of the most important conclusions of this thesis: music only becomes music in the mind of a human listener. There is no music without people.