HARMONY-ANALYSER.ORG - JAVA LIBRARY AND TOOLS FOR CHORDAL ANALYSIS

Ladislav Maršík

Dept. of Software Engineering, Faculty of Mathematics and Physics
Charles University, Malostranské nám. 25, Prague, Czech Republic
marsik@ksi.mff.cuni.cz

ABSTRACT

We present a new Java library and tools for analysis of music harmony focused on chords, chord progressions and chroma vectors. The underlying model is capable of creating, naming and analysing chords, as well as evaluating chord distances including Tonal Pitch Space, geometric distances on Tonnetz grid, or chord complexities. Special attention is given to the experimental distances between chroma vectors. Our system can take input in the form of text, MIDI input device, or the WAV file format. We provide tools for chordal analysis and for creating visual representations - chord segmentations or line graphs of chord or chroma distances. The system is extensible by creating additional plugins and provides an easy way of incorporating C++ based Vamp plugins, thus forming a suitable framework for music analysis in Java. Our *JHarmonyAnalyser* library and tools under *harmony-analyser.org* can be used for music analysis or as a feature extraction for retrieval tasks.

1. INTRODUCTION

Automatic chord estimation has been a major task in Music Information Retrieval (MIR) in the 21st century. Fujishima has first compared the chroma vectors to a chord dictionary and obtained chord progressions in 1999 [6]. Since then, effort have been put not only towards great variety of techniques for chord estimation [11], but also towards improving recent MIR tasks by analysing chroma or chord sequences. Notable examples are in cover song identification task, where using reduced chroma information, such as averaging chroma for every beat [5], or chroma landmarks [1], yields good results. Similarly for chord sequences, extracting and comparing fingerprints is a proven way to search for a cover song [8]. While comparing chroma or chord sequences looks to be a standard way for today's harmony analysis, new ideas have arisen in the past years, pioneering similarity and distances between chords themselves [4][13] and calling for further research on this topic. The paradigm shift is to look at the chord as a point in the space, and treat progression in musical piece as a path in this space.

Our work is following up on these efforts. We provide library and tools capable of creating, naming and analysing chords and visualize chord progressions. Furthermore, we gather all relevant chord distances and provide tools for their comparison. We also pioneer new distances between chroma vectors, offering multiple options to approach this novel concept.

The paper is structured as follows. In Section 2, we look at the known chord distances and form suggestions towards chroma distances. In Section 3, we show the contents of our *JHarmonyAnalyser* library and explain more in depth the new models we have developed (chord complexity distance, chroma vector distances). In Section 4 we present how new plugins can be created and added to the system, in order to achieve custom analysis. We wrap up by showing snapshots of GUI tools in Section 5, and our course for future work in Section 6.

2. CHORD AND CHROMA DISTANCES

In music theory, the concept of chord distance is long-known as *Tonal Pitch Space* (TPS), introduced by Lerdahl in 2001 [9] (Figure 1).

(a)	0												(0)
(b)	0						7						(0)
(c)	0			4			7						(0)
(d)	0		2		4	5		7		9		11	(0)
(e)	0	1	2	3	4	5	6	7	8	9	10	11	(0)

Fig. 1. Tonal Pitch Space model proposed by Lerdahl [9]. Chords are visualized in five levels, from *(a)* to *(e)*: octave, fifths, triadic, diatonic, and chromatic level. Depicted is the space of a *C major* chord in the *C major* key context. Distances between chords are then calculated as a sum of number of shifts on levels *(c)* and *(d)* and number of non-common pitch classes on levels *(a)* to *(e)* between the two chords. We refer the reader to [9] or [4] for more information.

Only recently, TPS was used to improve chord estimations by Rocher et al. [13], and music similarity tasks by De Haas et al. [4]. Notable models, distinct from TPS, are the ones based on Tonnetz grid (Figure 2), from which the most developed is Chew's spiral model [2]. There are also other ways to represent chord distances, out of the scope of our work, all summarized in the work of Rocher et al. [13].

In the literature, we often encounter comparisons of chroma-like structures and chord or key candidates. Distinct from chord-to-chord distance, in this use case, we first derive a so called *pitch class profile* from the musical piece or its part. An example can be averaging chroma vectors for the whole piece. Resulting structure is a vector of floats, similar to chroma vector. We then estimate the key for the musical piece, by comparing the pitch class profile to the profile of the key candidate [7]. A similar use case is that of estimating a chord for a given set of chroma vectors. Chord candidates are typically chosen from a suitable dictionary, such as 24 major and minor chords. Commonly used methods are Euclidean distance, or weighted sums [6], the latter being an inverted distance (higher the sum, closer is the queried vector to the chord candidate). In these examples we can see that distance measures are meaningful between chroma-like structures and chords. But we need to keep in mind that chroma vectors and chords occur on different levels: chroma vectors are obtained by a Fourier transform and represent a short frame of music, while chords can be obtained by processing chroma vectors and represent the changes in harmony.

While distances between adjacent chords in a musical piece have been proposed for improving MIR tasks [13], to the best of our knowledge, distances between two (adjacent) chroma vectors have not been extensively studied, although they can represent meaningful information. Above mentioned methods can be easily applied, however, it is questionable if they will be valid for transition between two short instants. More interestingly, musicology-based methods such as TPS can be employed to obtain chroma vector distances. We elaborate these new topics in our system.

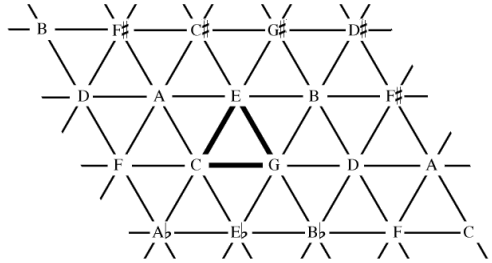

Fig. 2. Tonnetz grid as proposed by Riemann [12] in 19th century. Highlighted is the *C major* chord.

3. LIBRARY OVERVIEW

harmony-analyser.org is a home for *JHarmonyAnalyser* library as well *as* simple GUI tools to present its main functions, together forming a cohesive system for chordal analysis. It contains two main packages:

- *chord_analyser* package*,* containing Java classes for evaluating chord distances
- *chroma_analyser* package, containing Java classes for evaluating experimental chroma distances

Both packages have a similar structure, and fulfil these common requirements:

1. The packages contain base classes, that supply them with the representations of tones, intervals and scales (typically one Java class for each entity).
2. The packages contain the main classes, *Chord* and *Chroma* respectively for *chord_analyser* and *chroma_analyser* package*.* Various constructor inputs are supported, including text input.
3. The packages provide methods to derive distances between its main entities. For this purpose, multiple *model* classes are provided, representing some model of computation (e.g. TPS model).

To ease the work with a real-life input, *services* package is included, containing tools for reading WAV files, or interpreting MIDI input from external MIDI instrument (e.g. MIDI keyboard).

We show how all the packages work together in an example scenario: let us consider MIDI input keyboard plugged in to our machine. Classes available in *services* packages are able to get the keyboard and decode its input. We may use *chord_analyser* package to construct chord, or *chroma_analyser* package to construct chroma from the input on the keyboard. We choose to proceed with playing two chords (e.g. *C major* and *G7*) and creating two *Chord* objects. Then, using *TonalPitchSpace* model located in *chord_analyser,* we calculate distance between *C major* and *G7*, outputting result to a text area in GUI.

We explain the main contents of each package in the subsections below.

3.1. chord_analyser package

With *JHarmonyAnalyser* library being updated on a regular basis, new models are being included to the packages, to achieve an up-to-date selection. Brief summary of the current model classes contained in

chord_analyser package in version 1.2 of *JHarmonyAnalyser* are:

- *TonalPitchSpace* - as defined by Lerdahl [9], with symmetricity fix as described by De Haas et al. [4].
- *ChordComplexityDistance* (CCD) - defined in our previous work [10] as a complexity of transition between two subsequent chords, this experimental distance treats the added dissonant tones on the top of major or minor chord as additional layers of complexity. Rules are defined to add/remove/modify the dissonant tones. Resulting distance is defined as the complexity of constructing the next chord from the previous chord and is similar to edit distance of two strings. Figure 3 shows the distance between *C major* chord with the added *a* tone, and *C major* chord with the added *f#* tone.
- *Tonnetz* - implementation of the Tonnetz grid and distances of major and minor chords on the grid (see Figure 2).

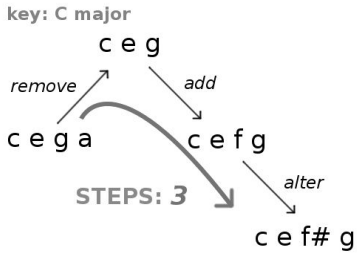

Fig. 3. Diagram showing the Chord Complexity Distance (CCD) between two chords. First, the *a* tone needs to be removed to obtain common ancestor for both chords, which is *C major* chord in a *C major* key. Afterwards, we add the *f* tone from the same key, and alter it to *f#* outside the key. Resulting complexity of the transition is 3. For more information we refer the reader to [10].

3.2. chroma_analyser package

In this package we provide classes to calculate experimental distances between chroma vectors. Example models in version 1.2 of the library are: *EuclideanDistance*, *WeightedSumDistance*, *SimpleDifference* and *ComplexityDifference*. As described in Section 2, Euclidean distance and weighted sums are commonly used for chord estimation. *chroma_analyser* package allows us to reuse these distances for two subsequent chroma vectors in the musical piece. In addition to these basic concepts, we offer *SimpleDifference sd(x,y)* for chroma vectors *x* and *y* as:

$$sd(x,y) = \sum_{i=1}^{12} |x_i - y_i|$$

Motivated by the levels present in Tonal Pitch Space (Figure 1) and chord complexity distance presented in Section 3.1, we propose *ComplexityDifference cd(x,y)* for chroma vectors *x* and *y* as:

$$cd(x,y) = \sum_{i=1}^{12} |w(x)_i x_i - w(y)_i y_i|$$

In this definition, *w(x)* and *w(y)* are weight vectors, which have their 12 values dependent on the context for the transition from *x* to *y*. To explain how to find the weights, we define *c(x)* as a function that returns chord estimate for a given chroma vector *x*, and *k(c, d)* as a function that returns common key for chords *c and d*. Both functions *k* and *c* have their return values in the form of boolean vectors of size 12 (for a chord estimate *c(x)* of a chroma vector *x*, $c(x)_i = true$ if the pitch class *i* belongs to *c(x)*, $c(x)_i = false$ otherwise). Usually for a vector returned by function *c*, 3 values will be *true* and for a vector returned by function *k*, 7 values will be *true*, which are the standard numbers of tones for chords and keys. As an example transition, we can imagine *x* and *y* both being estimated to *C major* chord ($c(x) = c(y) = C\ major$ chord) and *C major* being the common key ($k(c(x), c(y)) = C\ major$ key). Weight vector *w(x)* is defined as follows[1]:

$$w(x)_i = 0 \Leftrightarrow c(x)_i = true$$
$$w(x)_i = 1 \Leftrightarrow k(c(x), c(y))_i = true \wedge c(x)_i = false$$
$$w(x)_i = 2 \Leftrightarrow k(c(x), c(y))_i = false \wedge c(x)_i = false$$

In other words, using the weight vector *w(x)*, we suppress the changes in between chord tones ($w(x) = 0$), we take into account changes in between the non-chord tones from the key ($w(x) = 1$), and we assign highest weights to the non-chord and non-key tones ($w(x) = 2$), to emphasize the non-diatonic movements in music. This way, we can observe the complexity of movements in tonal music.

4. EXTENSIBLE PLUGINS

The system under *JHarmonyAnalyser* library can also act as an easy-to-use framework for creating plugins for music analysis in Java (not necessarily related to chords or chroma vectors). There are ready-made plugins available in the *plugins* package of the library. Custom plugins can be created simply by following the proposed structure.

[1] We can get *w(y)* by the same equation, only swapping chroma vectors *x* and *y*.

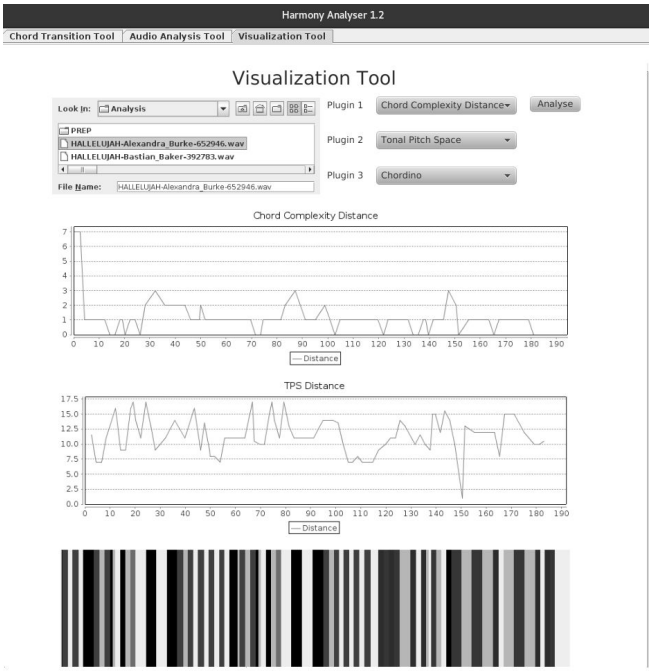

Fig. 4. Visualization Tool from *harmony-analyser.org*: selection of plugins and the resulting line graphs. Analysis of Hallelujah by Alexandra Burke is shown, containing results for both TPS and CCD distances on one screen. We can observe some chord distance peaks captured by both models, and some singled out by only by TPS or CCD distance. CCD peaks are around 30th, 90th and 150th second correspond to the characteristic chord progression for the song F A# C Dm A# C A7 Dmi.

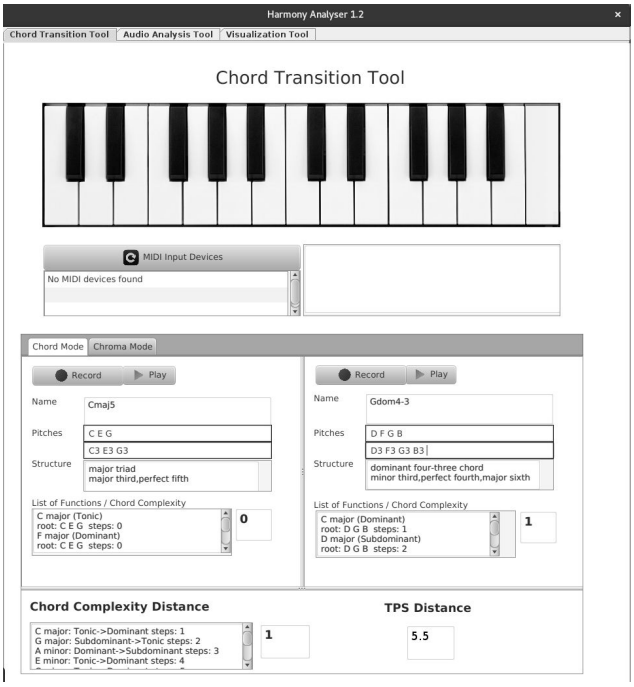

Fig. 5. Chord Transition Tool from *harmony-analyser.org*: capturing the MIDI input and outputting chord labels and distance information. *C major* and *G7* chords were captured on a piano keyboard.

Plugins form one of the core parts of the system, since they are responsible for opening and analysing WAV files using the models from the rest of the library. While distances between chords and chroma vectors can be calculated simply by calling a method from the model class, analysis of WAV file format, usually, takes more subsequent steps in order to achieve the desired result. For example, we first need to derive chroma vectors from the WAV file, and then we need to get distances between subsequent chroma vectors. Optionally, we may want to plot a line graph of the resulting distances. The resulting 3-step process would be unfit for one wrapper method, especially if we want to observe the partial results. Custom and ready-made plugins serve this purpose of layered analysis.

Inspired by GNU GPL licenced C++ Vamp plugins (*http://vamp-plugins.org*), a plugin has its own unique key and analyses the whole WAV input in its specific manner, producing results that are well documented. In fact, thanks to Java wrappers provided by Queen Mary University, we include a selection of Java-wrapped Vamp plugins in the *plugins/vamp_plugins* package. Differently from Vamp plugins, results from one plugin are very often taken by another plugin to continue with higher-level analysis (chroma vectors picked up by chord estimator, chord progression picked up by chord distance plugin). Each plugin needs to meet the following requirements:

1. It needs to have a *pluginKey*, a unique key for the plugin, e.g. *chord_analyser:tps_distance.*
2. It needs to provide a list of input files and their extensions.
3. It needs to provide a list of output files and their extension.
4. It needs to implement *analyse* method, which takes all the input files, and creates correct output files.

Example plugins are (for full list, please refer to the documentation on *harmony-analyser.org*):

- *chord_analyser:tps_distance* implementing Tonal Pitch Space model analysis for a WAV file.
- *chord_analyser:chord_complexity_distance* implementing CCD analysis for a WAV file.
- *chroma_analyser:complexity_difference* implementing chroma vector distance analysis by *ComplexityDifference* model for a WAV file.

Fig. 6. Audio Analysis Tool from *harmony-analyser.org*: selecting a folder with WAV files and choosing a desired plugin for analysis (Vamp plugins, Chord Analyser, Chroma Analyser or Post Processing).

5. GUI TOOLS

Along with the library and plugins, *harmony-analyser.org* also provides simple GUI tools (jointly named *harmony-analyser*), developed using *java.awt* package, to show the main library functions. Screenshots of the main tools are on Figures 4, 5 and 6. Example tools are:

- *Chord Transition Tool* - using MIDI or text input to analyse chords or chroma vectors.
- *Audio Analysis Tool* - batch analysis using selected plugins for an input folder with WAV files.
- *Visualization Tool* - plotting graphs and diagrams with chord or chroma vector distances for a WAV file.

6. CONCLUSION AND FUTURE WORK

We provide a maintained, tested and publicly available Java library and tools for chordal analysis, under *harmony-analyser.org* domain. *JHarmonyAnalyser* library can be used for feature extraction, visual analysis of musical pieces, or comparison of chord and chroma distances, as well as a simple framework for development of music analysis software in Java. As a future work we will focus on comparing the proposed distances, and obtaining results for MIR tasks such as cover song identification. We plan to add new models to the *chord_analyser* package, starting with the model proposed by Elaine Chew [2]. We will also broaden the plugin base with plugins and smoothing filters. Our other goal is the ease of installation (Linux) and availability in Java repositories (Maven). By providing these free tools we hope to encourage future research in the chordal analysis field.

7. ACKNOWLEDGMENTS

The study was supported by the Charles University in Prague, project GA UK No. 708314.

8. REFERENCES

[1] Bertin-Mahieux, T. and Ellis, D. P. W. “Large-Scale Cover Song Recognition Using Hashed Chroma Landmarks” In: *IEEE Workshop on the Applications of Signal Processing to Audio and Acoustics* (WASPAA 2011), 2011

[2] Chew, E. *Mathematical and Computational Modeling of Tonality.* Springer US, 2014

[3] De Haas, W. B., Magalhães, J. P., and Wiering, F. “Improving
Audio Chord Transcription by Exploiting Harmonic and Metric Knowledge” In: *Proceedings of the 13th International Society for Music Information Retrieval Conference* (ISMIR 2012), 2012

[4] De Haas, W. B., Veltkamp, R. and Wiering, F. *Tonal Pitch Step Distance:* “A Similarity Measure for Chord Progressions” In: *Proceedings of the 9th International Conference on Music Information Retrieval* (ISMIR 2008), 2008

[5] Ellis, D. P. W., and G. E. Poliner. “Identifying ‘Cover Songs’ with Chroma Features and Dynamic Programming Beat Tracking“ In: *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing* (ICASSP 2007), 2007

[6] Fujishima, T., “Realtime Chord Recognition of Musical Sound: A System Using Common Lisp Music“ In: *Proceedings of the International Computer Music Conference* (ICMC 1999), 1999

[7] Gómez, E. *Tonal Description of Music Audio Signals.* PhD thesis, Universitat Pompeu Fabra, 2006

[8] Khadkevich, M. and Omologo, M. “Large-Scale Cover Song Identification Using Chord Profiles” In: *Proceedings of the 14th International Society for Music Information Retrieval Conference* (ISMIR 2013), 2013.

[9] Lerdahl, F. *Tonal Pitch Space*. Oxford University Press, Oxford, 2001

[10] Marsik, L., Pokorny, J. and Ilcik, M. “Towards a Harmonic Complexity of Musical Pieces” In: *Proceedings of the 14th Annual International Workshop on Databases, Texts, Specifications and Objects* (DATESO 2014), CEUR Workshop Proceedings, vol. 1139, 2014

[11] McVicar, M., Santos-Rodriguez, R., Ni, Y. and Bie, T. D., "Automatic Chord Estimation from Audio A Review of the State of the Art" In: *IEEE/ACM Trans. Audio, Speech & Language Processing* 22 (2), pp. 556–575, 2014

[12] Riemann, H. *Dictionary of Music*. Augener Ltd., London, 1896

[13] Rocher, T., Robine, M., Hanna, P. and Desainte-Catherine, M. "A Survey of Chord Distances With Comparison For Chord Analysis" In: *Proceedings of the International Computer Music Conference* (ICMC 2010), 2010