# Chordal Analysis

Anna Selonick
Northwestern University
abselonick@gmail.com

Salil Gupta
Northwestern University
salil.gupta323@gmail.com

Nikita Patel
Northwestern University
nikipatel@u.northwestern.edu

## ABSTRACT
In this paper, we describe an algorithm to automatically identify harmonic chords based on Pardo and Birmingham's "Algorithm's for Chordal Analysis". Python is used to implement the algorithm and a library from github called python-midi is used to read in midi files. This work is based on the assumptions that the music being analyzed is tonal. The algorithm effectively segments and labels a piece of music to reflect the chordal structure of the composition. The algorithm was tested on 47 musical excerpts from *Tonal Harmony* by Stefan Kostka and Dorothy Payne.

## Categories and Subject Descriptors
G.1.6 Analyze.

## General Terms
Algorithms, Performance, Verification.

## Keywords
Chordal Analysis, Machine Perception of Music, Chord Identification, Harmonic Analysis, Music Segmentation.

## 1. INTRODUCTION
In the study of classical western music, harmonic analysis is an important part of understanding a piece. Identifying tonal centers and chords are equivalent to being able to understand the meaning of a sentence. This is a skill that musicians take years mastering, which makes creating an algorithm for it even more difficult. Noticing subtle cues in the music while knowing when notes are and are not important is part of a musician's challenge. This is what makes creating an algorithm to identify tonal centers and chords an interesting problem. The study takes a task usually done by musicians through qualitative analysis and turns it into a completely objectified process. This could be useful in many scenarios where one would want to quickly get the chord structure of a piece of music. For example, a classical musician may need to identify chords and tonal structure of a piece in order to play it and understand it better. This algorithm will save the musician time and allow them to skip chord identification and move straight to understanding how parts of the piece fit together. Another case could be if someone wanted to convert a pop song into guitar tabs but doesn't know how to read music. This would allow them to take the first step towards transforming a notated piece of music into something they understand.

Previous work has been done on harmonic analysis in the computer music research community [Smoliar 1980; Laden and Keefe 1989; Smaill et al. 1993]. This work has dealt with identifying the chord/tonal label for segments of music that are already known to belong to a single label. While these researchers only looked at pre-segmented input, Temperly and Sleator (1999) and Pardo and Birmingham (2001) attempt to address this problem. Temperly and Sleator try to segment the piece by identifying where the correct chord changes are, but only return the root name. Pardo and Birmingham are able to fairly successfully segment music and return the root name and chord quality.

We present an algorithm for chordal analysis that uses the HarmAn segmentation method as discussed in Pardo and Birmingham's work. Our algorithm will return the root name and quality of chord. Beyond what Pardo and Birmingham do, our algorithm will attempt to take musical structures such as passing tones and grace notes into account when performing the chordal analysis.

## 2. ALGORITHM
The algorithm we use in this study is based on the HarmAn segmentation algorithm. This section describes the two main functions of the algorithm, segmentation and labeling.

The system input is midi files. The midi files being tested are short components of longer classical compositions. The system will read in these midi files and then determine the segments that best split the audio based on harmonic changes. The system will then label each of these segments with a score of how similar that segment is to a chord template from a predefined dictionary. The system will then output the segments and chords that have the highest scores.

### 2.1 Midi Files
The software package we use is python-midi by github user vishnubob. The package allows users to read in midi files as well as create new midi files. Midi files that are read in are stored in a pattern object. The pattern object consists of multiple tracks, but in our case just one track. Each track consists of multiple events that are chronologically ordered by tick. A tick is a time representation of half a millisecond. Each event stores the pitch, velocity, and tick. It is also important to note that some midi files only contain note on events, where a velocity of zero functionally means note off.

### 2.2 Chord Templates
In tonal musical notation, each note in the chromatic scale and its twelve pitch classes are represented with integer notation and modulo-12 arithmetic. The pitch classes and integer names are shown in Figure 1. We created 72 templates to represent 6 base chords each of the 12 keys in classical western music. The templates contain either three or four integers, and represent a major, minor, diminished, dominant seventh, half diminished seventh, or fully diminished seventh chord. The base chords (templates written in C major) can be seen in Table 1.
To match a chord in a different key to one of our 72 templates, you take the modulo-twelve of the chord. For example, consider the three-note chord with integer values <24, 28, 31>. This chord modulo 12 is <0,4,7>, the base template for C major. Each template is stored in a dictionary along with the probability of it occurring (used for a tie breaking mechanism later) and the chord name, as shown in Table 1.

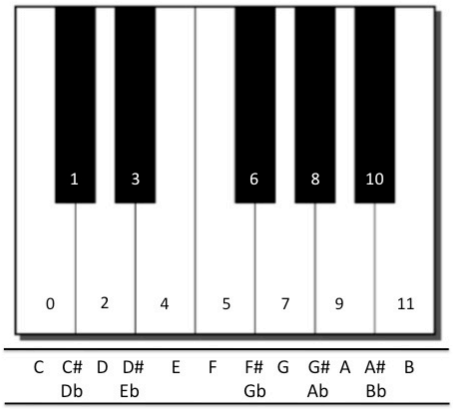


**Figure 1. Integer names for each of the 12 pitch classes in the western music scale.**

**Table 1. Base templates (for C major)**

| Chord Name | Template | Probability |
|---|---|---|
| Major Triad | <0 4 7> | 43.6% |
| Minor Triad | <0 3 7> | 19.4% |
| Dominant 7th | <0 4 7 10> | 21.9% |
| Fully Diminished 7th | <0 3 6 9> | 4.4% |
| Half Diminished 7th | <0 3 6 10> | 3.7% |
| Diminished Triad | <0 3 6> | 1.6% |

## 2.3 Segmentation and Directed Acyclic Graph

This first part of the algorithm deals with segmentation of the music. While segmentation and labeling is actually done together, it makes the most sense to discuss the segmentation process first.

A *partition* is a point in time where there is harmonic change in the music, which is when one or more notes change. In a midi file, this is marked by a note on or note off event. A *minimal segment* is everything in between two consecutive partition points. These two structures can be seen in Figure 2.

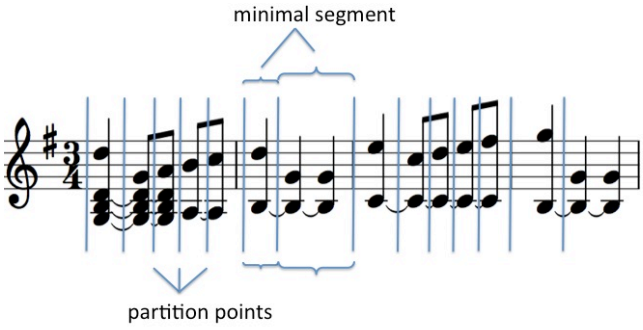


**Figure 2. Partition points and minimal segments in an excerpt from our dataset.**



Each minimal segment acts as a node and contains the start tick, an end tick, and an array of all the events that happen in between the ticks. The minimal segments serve as nodes in a directed acyclic graph (DAG). As shown in Figure 3, each of these minimal segments is connected to every other minimal segment that comes after it. Every *segment* is represented by an edge in the DAG, and is made up of any number of consecutive minimal segments. In order to determine the correct labeling for the piece of music, the highest cost path of the DAG is found using the HarmAn segmentation algorithm as described in section 2.5. Edge scores represent the correlation between the chord name and the notes in the segment.

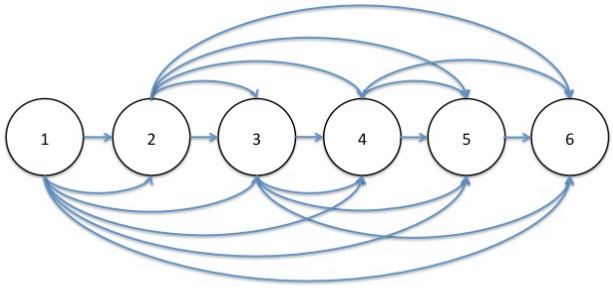


**Figure 3. DAG – Directed Acyclic Graph used in this algorithm.**

## 2.4 Edge Scoring

The scoring process for each edge, or segment, begins with calculating a weight for each note that appears in that segment. These note weights are determined by counting the number of minimal segments that the note appears in the given segment. The same note in different octaves will each be given a separate weight.

To account for slight inaccuracies in the way midi files were written and musical structures like passing tones and grace notes, we also factored in minimal segment length into note weights. For each piece of music, the smallest minimal segment length was recorded. For any note in a minimal segment equal to this recorded length, only 0.25 is added to the weight instead of 1.

The second step in scoring an edge is to give a score to each template based on note weights. For each individual template, the score is calculated using this formula:

$$S = P - (M + N)$$

Where S is the score, P is positive evidence, N is negative evidence, and M is misses. P is calculated by adding the weights of all the notes in the segment that are also in the template. N is calculated by adding the weights of all the notes in the segment that are not found in the template. M is the number of notes in the template that are not matched by any note in the segment. While calculating P and N, octave does not matter, so the note mod12 is compared to the elements of the template.

Once every template has a score for the given segment, the highest score is assigned to the edge. In the event of a tie, we have two tie breaking mechanisms. The first takes into account the probabilities discussed before. This is essentially choosing the template for the chord that is most likely to appear in classical music (the dataset we are using for testing). The second tiebreaker looks at the root note of each template (the first element of a template). When deciding between two templates, the template with a higher weight for the root note will be chosen.

## 2.5 Labeling

After creating an edge matrix to represent the DAG of all possible chords with their scores, we found the longest path in this DAG. By its construction, the DAG is already topologically sorted. We can then just traverse the vertices in topological order. For each vertex u, we check each of its adjacent vertices v to see if the distance to u plus the weight of edge (u,v) is greater than the current distance to v. If, so we change the distance associated with v to be the distance to u plus the weight of edge (u,v). We add (u,v) to the longest path, so we can retrace it later on.

# 3. ANALYSIS

The dataset used to test our algorithm is 47 excerpts from *Tonal Harmony* by Stefan Kostka and Dorothy Payne (the same ones used in Prof. Pardo's work described in "Algorithms for Chordal Analysis"). These excerpts have been written as midi files, with lyrics that reflect the correct chords that the algorithm should produce.

## 3.1 Performance Measure

The performance of the algorithm was determined by two factors: how similar the chord chosen was to the real chord and how similar the segmentation was. In order to determine similarity we used a Euclidean distance metric. For each chord label returned by the algorithm we performed a Euclidean distance between that chord and its segment time and each chord and segment time in the answer. We then normalized the two distance measures and added them together. This then gave us a penalty, which we subtracted from one to give us a score for the piece.

When we came across labels that we did not account for, such as X Ger6, we gave a distance measure of eleven for the similarity of the chords. Eleven was chosen because this was calculated to be the average distance measure found for the Euclidean distance of all the chords, and we assumed that our algorithm performs at the average. Also, when a label returned by the algorithm was a 3-note chord while the label given by the answer key was a 4-note chord, or vice versa, we added three to the distance measure of the chords. Three was chosen because if we assume that each note in a 4-note chord provides a quarter of the penalty we can then use a quarter of the average distance measure, which is three.

## 3.2 Results

We found the average penalty was 0.05 where the minimum possible penalty is 0 and the maximum penalty is 0.20. The standard deviation of the penalties was 0.037 while the median of the data is 0.049. The distribution of the data can be seen in figure 4. The outlier in the box and whisker plot can be ignored because the answer key for that excerpt was incomplete.

Excerpts that scored well with our algorithm tended not to have chords that we did not account for in our list of templates (i.e. X_Ger6). Excerpts also did better if they had a clearer transition between different chords and if the notes within a segment very clearly matched a single template.

We also found that taking into account passing tones and grace notes did better than not taking them into account, although marginally. We conducted a 2-sample t-test and found that the data does not reject the null hypothesis, or there is no difference between accounting for passing tones and grace notes and not accounting for them. We cannot reject the null hypothesis because the p-value is 0.116, which is greater than 0.05. This outcome is probably due to the fact that our sample size is too small.

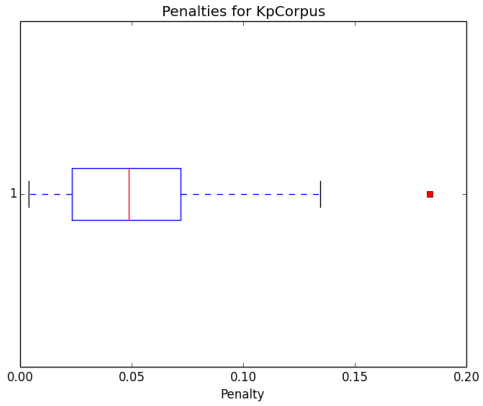


**Figure 4. Plot of penalties when performance analysis was done on the dataset from *Tonal Harmony***

# 4. CONCLUSION

An algorithm was presented for automatically performing chordal analysis on an excerpt of tonal music. Based off of pervious research and the HarmAn segmentation algorithm, this algorithm also attempts to handle musical structures such as passing tones and grace notes. The algorithm can identify six base chord types, and is based on the assumption that the music being analyzed is tonal. In contrast to most other methods, segmentation of the music and labeling of chords is done simultaneously. The method employs two tie-breaking methods. Finally, further work can be done to improve the accuracy of the segmentation algorithm.

# 5. ACKNOWLEDGMENTS

Our thanks to Professor Brian Pardo of the Northwestern University EECS Department.

# 6. REFERENCES

[1] Laden, B., and D. H. Keefe. 1989. *The Representation of Pitch in a Neural Net Model of Chord Classification.* Computer Music Journal 13(4):12–26. http://www.jstor.org/discover/10.2307/3679550?sid=21105408814671&uid=4&uid=2.

[2] Pardo, B., and W. P. Birmingham. 2001. *Algorithms for Chordal Analysis.* Ann Arbor, Michigan: Department of Electrical Engineering and Computer Science, The University of Michigan. http://www.cs.northwestern.edu/~pardo/courses/eecs352/papers/chord labeling midi - pardo.pdf.

[3] Smaill, A., et al. 1993. *Hierarchical Music Representation for Composition and Analysis.* Computers and the Humanities 27:7–17. http://link.springer.com/article/10.1007/BF01830712.

[4] Smoliar, S. 1980. *A Computer Aid for Schenkerian Analysis.* Computer Music Journal 4(2):41–59. http://www.jstor.org/discover/10.2307/3680082?sid=21105408756791&uid=4&uid=2.

[5] Temperley, D., and D. Sleator. 1999. *Modeling Meter and Harmony: A Preference-Rule Approach.* Computer Music Journal 23(1):10–27. http://dl.acm.org/citation.cfm?id=1116140.